Data Mining and Exploration of the Nuclear Science References

by Andrew Valencik

A Thesis Submitted to Saint Mary's University, Halifax, Nova Scotia in Partial Fulfillment of the Requirements for the Degree of Master in Applied Science.

December 2015, Halifax, Nova Scotia

Approved:	Dr. Roby Austin Supervisor
Approved:	Dr. Adam Sarty Examiner
Approved:	Dr. Paul Muir Examiner
Approved:	Dr. Pawan Lingras Examiner
Approved:	Dr. Svetlana Barkanova External Examiner

December 11th, 2015

Abstract

Data Mining and Exploration of the Nuclear Science References

by Andrew Valencik

The Nuclear Science References (NSR) is a carefully curated bibliographic dataset focused on nuclear science literature. A domain-specific search engine and supporting tools have been developed to aid and encourage the exploration of the NSR. User queries are analyzed to form a series of filters to retrieve relevant NSR entries from a database. The resulting information is presented in multiple views including lists, bar charts, and network graphs. The network graph representations offer unique insights on collaborations centered around a given parameter such as a nuclide or group of authors. The capability of clustering algorithms to expose trends within the dataset is demonstrated by clustering authors based on publication traits. A vector space model based on the metadata provided in the NSR is used to recommend semantically similar NSR entries. The completed work serves as both an example and a framework for future analysis of the NSR.

December 11th, 2015

Acknowledgments

There is no one more deserving of thanks than my very dedicated supervisor, Dr. Roby Austin (Saint Mary's University - Dept. of Astronomy and Physics). I have had the great advantage of her endless support throughout my undergraduate degree and my masters. I am not sure if it was my curiosity or hers that was so contagious, but her indulgence has resulted in so much encouragement and a wonderful picture of how science should work. Thank you Roby, for your considerable and consistent efforts in correcting, suggesting, motivating, and inspiring.

I would also like to thank my committee members, Dr. Adam J. Sarty (Saint Mary's University - Dept. of Astronomy and Physics), Dr. Paul Muir (Saint Mary's University - Dept. of Mathematics and Computing Science), Dr. Pawan Lingras (Saint Mary's University - Dept. of Mathematics and Computing Science), and my external examiner Dr. Svetlana Barkanova (Acadia University - Dept. of Physics). Their support over the past 3 years has been critical to completing this work.

My partner in crime and life, Alicia Beazley, has provided so much support when I needed it most. Thank you for your understanding, reassurance, and reminders that I actually need to eat and sleep.

I appreciate so much my family's support. I am very happy I have made them proud.

Many thanks to the Mathematics and Computing Science department which has provided a very valuable collection of minds that were a constant source of suggestions and support. Finally, I would like to acknowledge that I cannot thank everyone who has helped of the last 3 years because there are so many of you, and for that I am very grateful.

Contents

List of Figures

List of Code Snippets

List of Tables

1 Introduction

Information retrieval has been repeatedly improved by large search engines like Google, Yahoo, DuckDuckGo, and more. Vast quantities of information are now easily retrieved on an extensive array of subjects. Scientific literature has received special attention through projects like Google Scholar or Microsoft Academic Search. These projects are generalized to accommodate all sciences. Information retrieval and data exploration can be improved by customizing an application to a specific domain.

The United States National Nuclear Data Center (NNDC) prepares an evaluated database of nuclear science literature that poses a rich opportunity for knowledge discovery directed at scientific work and study. The academic field of nuclear science is over one hundred years old, starting with the discovery of radiation [1]. This discovery represents the first of many entries in the Nuclear Science References database, collected, cataloged, distributed, and evaluated by the National Nuclear Data Center [2]. The Nuclear Science References, or NSR, has over 210,000 entries documenting the body of nuclear science literature, which provides the opportunity for knowledge discovery on the literature's metadata. The metadata that the NSR provides is contributed and maintained by neutral third party experts from the NNDC. This fact separates the information in the NSR from metadata available through services such as ResearchGate.

This work is a cross disciplinary effort, combining semantic information of nuclear physics literature and data mining techniques to build a custom application for data exploration and information retrieval in nuclear science. The practice of knowledge discovery and data mining on the NSR dataset can reveal trends in the collective scientific study of nuclear structure, processes, and detection. These data are presented through a web application that extends the existing facilities of the Nuclear Science References web retrieval system [3]. The ultimate goal of this work is to enable further analysis on the body of nuclear science literature.

This is a thesis that applies science from one domain to science from another domain. To facilitate understanding by all readers, no matter their expertise, background information is placed in the thesis close to where it is used. Thus, this thesis has no “theory” chapter, as essential knowledge is provided in the chapters that require it.

The primary contribution of this thesis is the construction of tools and a framework for future analysis into the valuable NSR dataset. Assertions about the interpretation of results from these tools have been avoided. The goal of this work was not to study physics or physicists, but instead to apply expertise in nuclear science and data analytics to enable diverse researchers, including network analysts and social scientists to explore the NSR dataset. The resulting work is flexible enough, by design, to be easily adapted to the needs of anyone investigating the NSR.

1.1 Thesis Organization

The existing NSR website and interface are outlined in this introduction chapter. Additionally, the web application created as a result of this work is discussed.

The transformation of the provided raw data is discussed in the Data Preparation and Data Representation sections. The data are stored in a database as discussed in The Database.

Chapter 3 is the first exploration of the NSR data and the database created with our data representation. We discuss the types of queries that can be made on the data, show some examples, and discuss the results. The Author Contributions section analyzes how the authors of works catalogued in the NSR contribute to the NSR collection as a whole. Visualizations for the queries discussed in this chapter are shown.

Chapter 4 introduces the concept and tools for analyzing the NSR data as a network graph. A brief overview of graph theory concepts and terminology is given. We discuss different queries that can be used to produce graphs of the NSR data in the Data Graphs and Nuclide Graphs sections. The Libraries used are discussed as well as an exportation feature.

Chapter 5 details the usage of tools and techniques from text mining. A paper recommender system is built in the Similar Papers section by using a modification of cosine similarity. The second component of this chapter analyzes author names to build a system of finding authors who may have multiple identifications in the NSR.

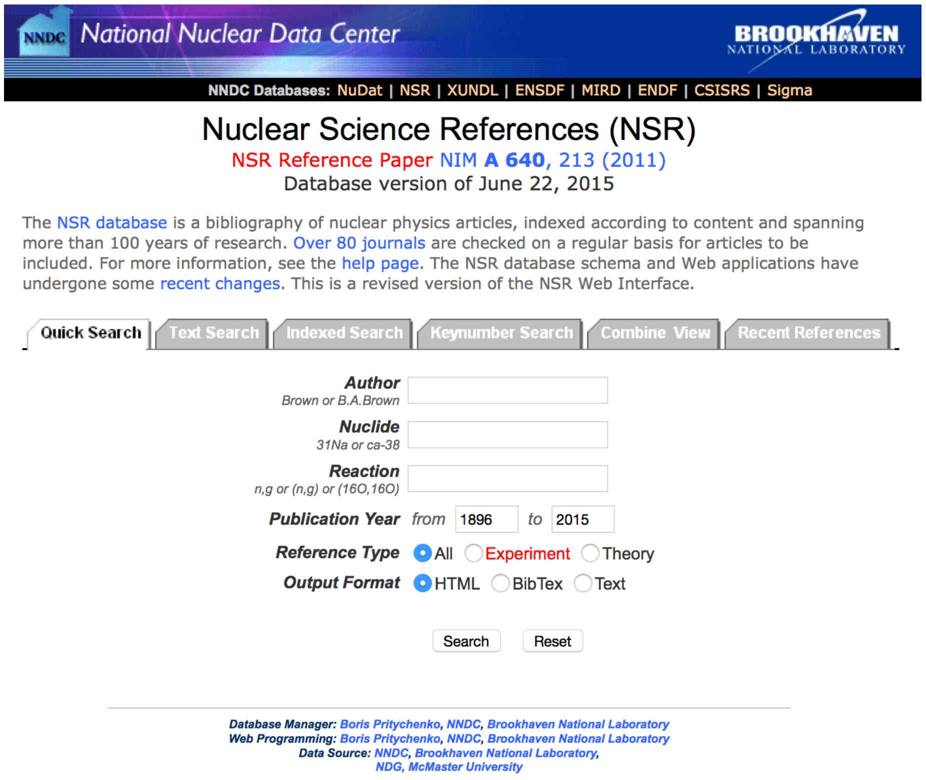
Chapter 6 discusses two common data mining techniques and their applications with regard to the NSR data. Association mining is performed on nuclides in papers, author names in papers, and author names in selectors. In the Cluster Analysis section, K-means is used to cluster authors.

Finally, we review the contributions and discuss future works in the Conclusions chapter.

1.2 The Nuclear Science References Website

Please note, the data that these interfaces interact with and return as results are fully discussed in the Data and Database section.

The NNDC maintains the NSR website which serves a web interface to the Nuclear Science References database. The functionality and architecture of the NSR database and web site is discussed by Pritychenko in [3]. Four search interfaces are offered: quick search, text search, indexed search, and keynumber search. The quick search interface is shown in Figure 1.


NNDC National Nuclear Data Center
BROOKHAVEN NATIONAL LABORATORY
NNDC Databases: NuDat | NSR | XUNDL | ENSDF | MIRD | ENDF | CSISRS | Sigma

Nuclear Science References (NSR)

NSR Reference Paper NIM **A 640**, 213 (2011)

Database version of June 22, 2015

The NSR database is a bibliography of nuclear physics articles, indexed according to content and spanning more than 100 years of research. Over 80 journals are checked on a regular basis for articles to be included. For more information, see the help page. The NSR database schema and Web applications have undergone some recent changes. This is a revised version of the NSR Web Interface.

Author *Brown or B.A.Brown*	
Nuclide *31Na or ca-38*	
Reaction *n,g or (n,g) or (16O,16O)*	
Publication Year	*from* 1896 *to* 2015
Reference Type	[✓ All] Experiment Theory
Output Format	[✓ HTML] BibTex Text

Database Manager: *Boris Pritychenko, NNDC, Brookhaven National Laboratory*
Web Programming: *Boris Pritychenko, NNDC, Brookhaven National Laboratory*
Data Source: *NNDC, Brookhaven National Laboratory, NDG, McMaster University*

Figure 1: The main interface for the NNDC's NSR website. Captured June 28th 2015

The quick search functionality is the most commonly used interface [3]. It enables searching by author name, nuclide, or reaction. Two types of filters are available to limit the results: a year range, and reference type which can return only experimental or theory entries. Each of the search fields show examples of the type of search as well. For example the author field shows a search for an author using their first initials and their last name, or only their last name.

The text search interface enables text searching in the title, keywords, or both fields. The search is not case sensitive and requires a search string of at least three characters in length. Phrases can be used by enclosing them in quotes. The user can specify a publication year range, or choose a date to filter when the entries were added to the database. Additionally the user can enable 'primary only' or 'require measured quantity' flags. The results can be sorted in ascending or descending order and presented in HTML, BibTex, Text, Keynum, or Exchange formats. The quick search results do not offer these output customizations.

The interface for indexed searching is similar to the Text Search. The most important difference is the functionality offered by the browse buttons for the search parameters. The user can select a parameter of the following types: Author, FirstAuthor, Nuclide, Target, Parent, Daughter, Subject, Measured, Deduced, Calculated, Reaction, Incident, Outgoing, Journal, Topic, Z(range). For each of the types available the browser button will redirect to another page that either details the possible values or provides another search through the possible values. For example, Author and First Author direct to a simple search interface that allows some partial matches against the list of known authors.

Search queries are remembered and presented in the 'Combine View' tab. Users can combine the results of recent queries with boolean logic. Analysis is offered on the search queries which displays how many nuclides, authors, journals, and publication years the query involved.

1.3 NSR Explorer - Web Application

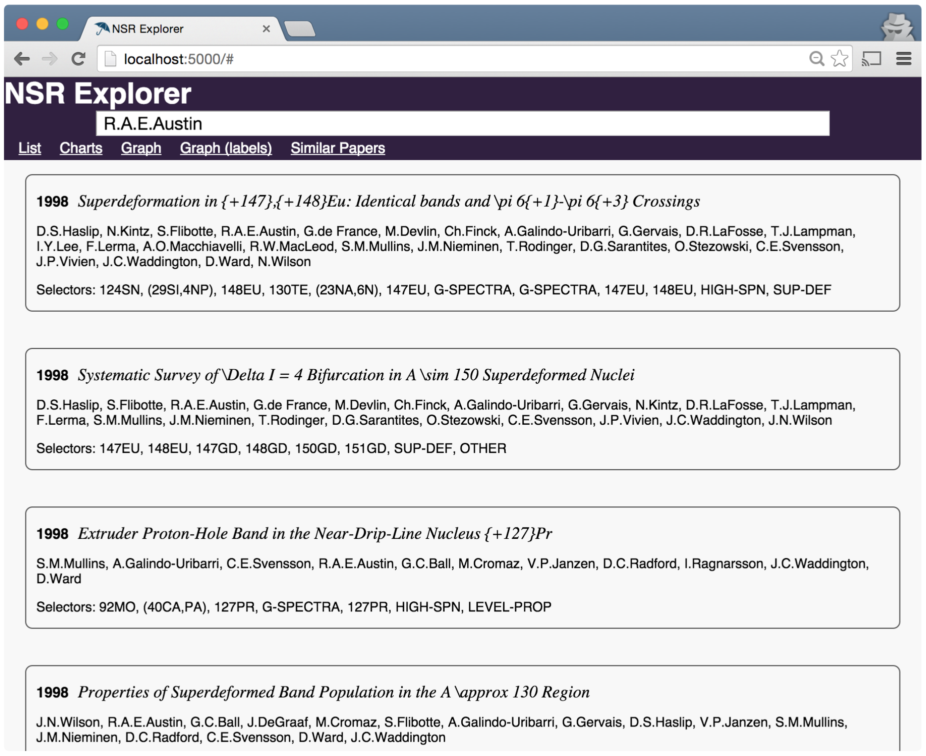
As a contribution to this thesis, we have developed a web application, NSR Explorer, to increase accessibility to exploration of the Nuclear Science References data. This includes the authors documented, the entries recorded and keyworded, their links, and all available metadata for the nearly 120 years of records. The application makes use of a web interface to aid in increasing accessibility. All that is required to use the application is a modern web browser. Interactive visualizations are used to encourage exploration of the data. Additionally, the new database structure that is developed in this work enables searches that were previously cumbersome or impossible.

The web application presents a single search interface as shown in Figure 2. This interface supports a variety of filters to retrieve NSR entries from the database. For example, if the user inputs a string that matches an author name, the application retrieves all NSR entries for that author. An example list view for input "R.A.E.Austin" is shown in Figure 2.

If the user inputs a year or year range such as "1989" or "1970-1979" the entries that were published in those years are retrieved. Selector values (further discussed in Keyword Abstracts) such as "11Li" can also be used as a filter. Finally, these filters can be combined to form advanced queries retrieving NSR entries on certain authors working on particular nuclides in a given range as shown in Figure 3.

The web application provides 5 basic views of retrieved NSR entries. The "List" view shows basic information such as the publication year, title, author list, and selector values. The "Charts" view shows a bar chart depicted the amount and type of NSR entry per year as shown in Figure 4. The "Graph" view shows a network graph with nodes coloured by author cluster membership as shown in Figure 5 (further discussion on author clusters in Cluster Analysis). The "Graph (labels)" view presents the same information as the "Graph" view but with text labels on the nodes as shown in 6. Analysis of the network graphs is further discussed in Network Analysis and Visualization.

The "Similar Papers" reuses the "List" view but instead displays semantically similar papers for the searched author. An example for author "R.A.E.Austin" is shown in Figure 7. The method by which NSR entries are determined to be similar is discussed in the Similar Papers section.



1998 *Superdeformation in {+147},{+148}Eu: Identical bands and \pi 6{+1}-\pi 6{+3} Crossings*

D.S.Haslip, N.Kintz, S.Flibotte, R.A.E.Austin, G.de France, M.Devlin, Ch.Finck, A.Galindo-Uribarri, G.Gervais, D.R.LaFosse, T.J.Lampman, I.Y.Lee, F.Lerma, A.O.Macchiavelli, R.W.MacLeod, S.M.Mullins, J.M.Nieminen, T.Rodinger, D.G.Sarantites, O.Stezowski, C.E.Svensson, J.P.Vivien, J.C.Waddington, D.Ward, N.Wilson

Selectors: 124SN, (29SI,4NP), 148EU, 130TE, (23NA,6N), 147EU, G-SPECTRA, G-SPECTRA, 147EU, 148EU, HIGH-SPN, SUP-DEF

1998 *Systematic Survey of \Delta I = 4 Bifurcation in A \sim 150 Superdeformed Nuclei*

D.S.Haslip, S.Flibotte, R.A.E.Austin, G.de France, M.Devlin, Ch.Finck, A.Galindo-Uribarri, G.Gervais, N.Kintz, D.R.LaFosse, T.J.Lampman, F.Lerma, S.M.Mullins, J.M.Nieminen, T.Rodinger, D.G.Sarantites, O.Stezowski, C.E.Svensson, J.P.Vivien, J.C.Waddington, J.N.Wilson

Selectors: 147EU, 148EU, 147GD, 148GD, 150GD, 151GD, SUP-DEF, OTHER

1998 *Extruder Proton-Hole Band in the Near-Drip-Line Nucleus {+127}Pr*

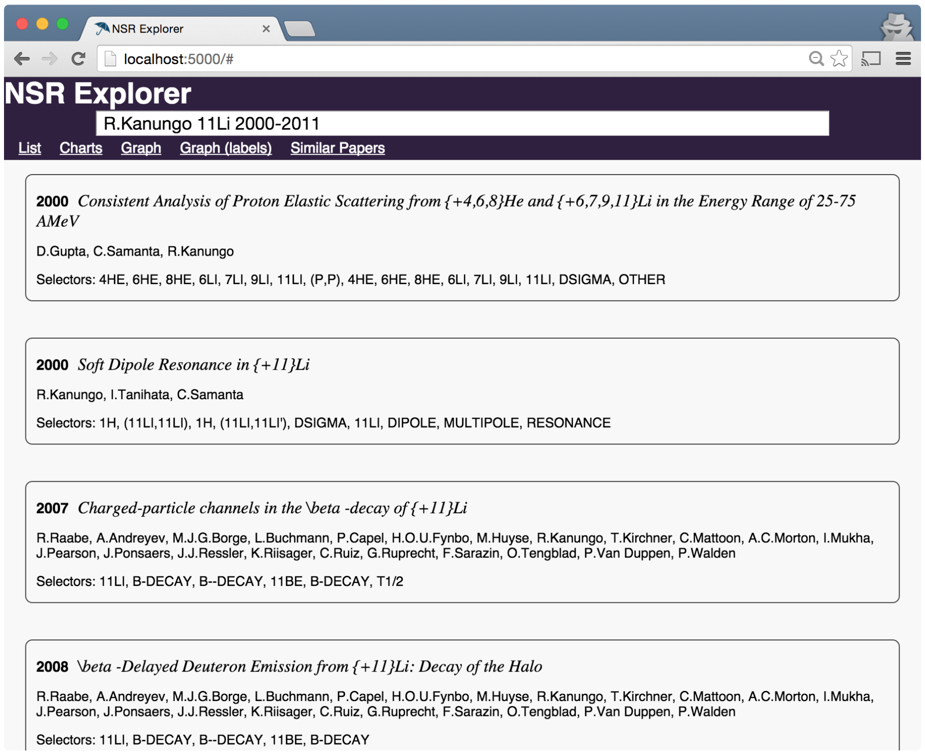
S.M.Mullins, A.Galindo-Uribarri, C.E.Svensson, R.A.E.Austin, G.C.Ball, M.Cromaz, V.P.Janzen, D.C.Radford, I.Ragnarsson, J.C.Waddington, D.Ward

Selectors: 92MO, (40CA,PA), 127PR, G-SPECTRA, 127PR, HIGH-SPN, LEVEL-PROP

1998 *Properties of Superdeformed Band Population in the A \approx 130 Region*

J.N.Wilson, R.A.E.Austin, G.C.Ball, J.DeGraaf, M.Cromaz, S.Flibotte, A.Galindo-Uribarri, G.Gervais, D.S.Haslip, V.P.Janzen, S.M.Mullins, J.M.Nieminen, D.C.Radford, C.E.Svensson, D.Ward, J.C.Waddington

Figure 2: The list view for "R.A.E.Austin"



NSR Explorer

R.Kanungo 11Li 2000-2011

2000 *Consistent Analysis of Proton Elastic Scattering from {+4,6,8}He and {+6,7,9,11}Li in the Energy Range of 25-75 AMeV*

D.Gupta, C.Samanta, R.Kanungo

Selectors: 4HE, 6HE, 8HE, 6LI, 7LI, 9LI, 11LI, (P,P), 4HE, 6HE, 8HE, 6LI, 7LI, 9LI, 11LI, DSIGMA, OTHER

2000 *Soft Dipole Resonance in {+11}Li*

R.Kanungo, I.Tanihata, C.Samanta

Selectors: 1H, (11LI,11LI), 1H, (11LI,11LI'), DSIGMA, 11LI, DIPOLE, MULTIPOLE, RESONANCE

2007 *Charged-particle channels in the \beta -decay of {+11}Li*

R.Raabe, A.Andreyev, M.J.G.Borge, L.Buchmann, P.Capel, H.O.U.Fynbo, M.Huyse, R.Kanungo, T.Kirchner, C.Mattoon, A.C.Morton, I.Mukha, J.Pearson, J.Ponsaers, J.J.Ressler, K.Riisager, C.Ruiz, G.Ruprecht, F.Sarazin, O.Tengblad, P.Van Duppen, P.Walden

Selectors: 11LI, B-DECAY, B--DECAY, 11BE, B-DECAY, T1/2

2008 *\beta -Delayed Deuteron Emission from {+11}Li: Decay of the Halo*

R.Raabe, A.Andreyev, M.J.G.Borge, L.Buchmann, P.Capel, H.O.U.Fynbo, M.Huyse, R.Kanungo, T.Kirchner, C.Mattoon, A.C.Morton, I.Mukha, J.Pearson, J.Ponsaers, J.J.Ressler, K.Riisager, C.Ruiz, G.Ruprecht, F.Sarazin, O.Tengblad, P.Van Duppen, P.Walden

Selectors: 11LI, B-DECAY, B--DECAY, 11BE, B-DECAY

Figure 3: The list view of a query taking advantage of combined filters

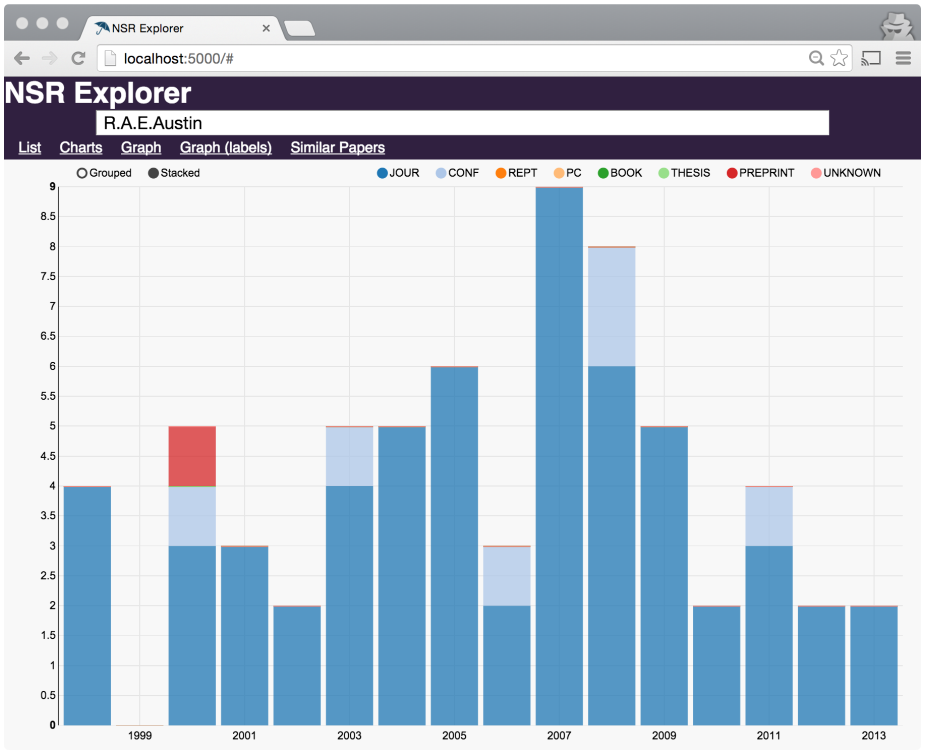


Figure 4: The chart view for “R.A.E.Austin”

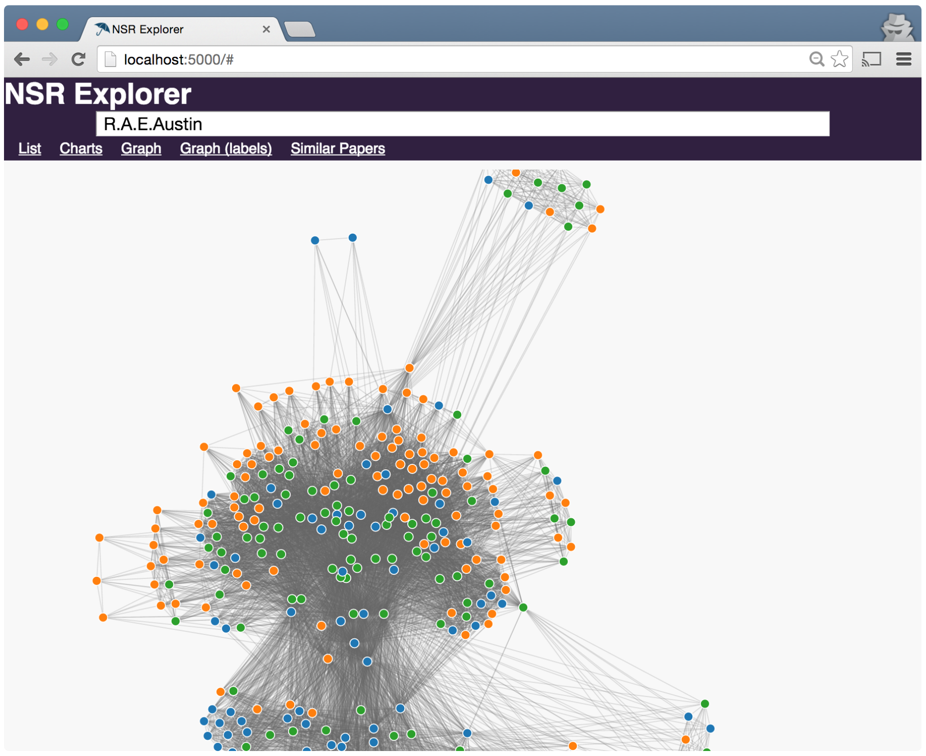


Figure 5: The graph view for “R.A.E.Austin”

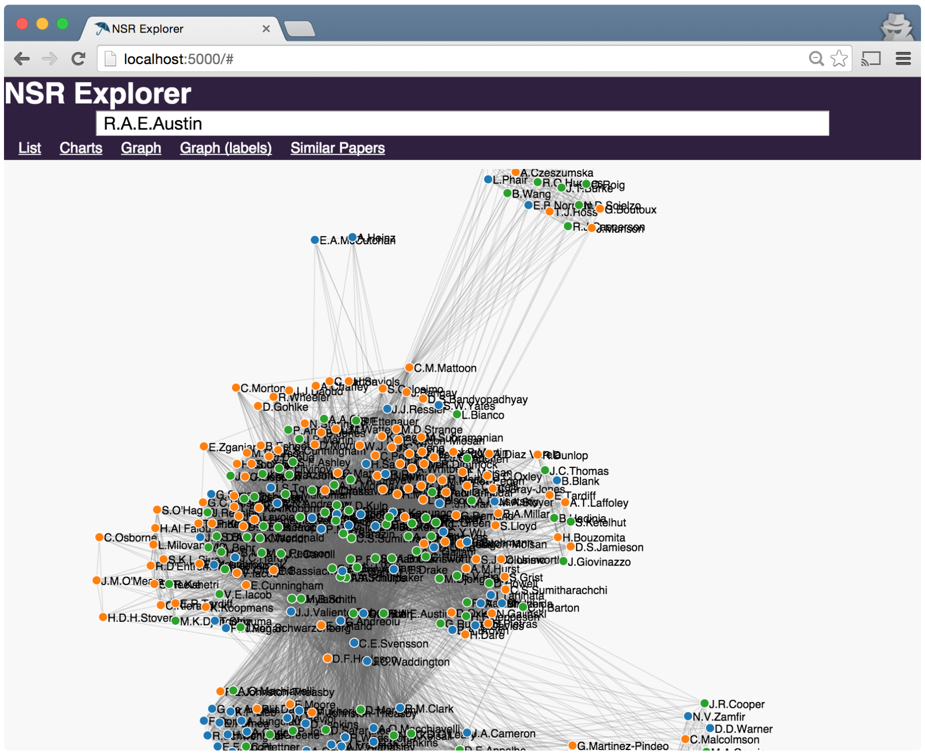
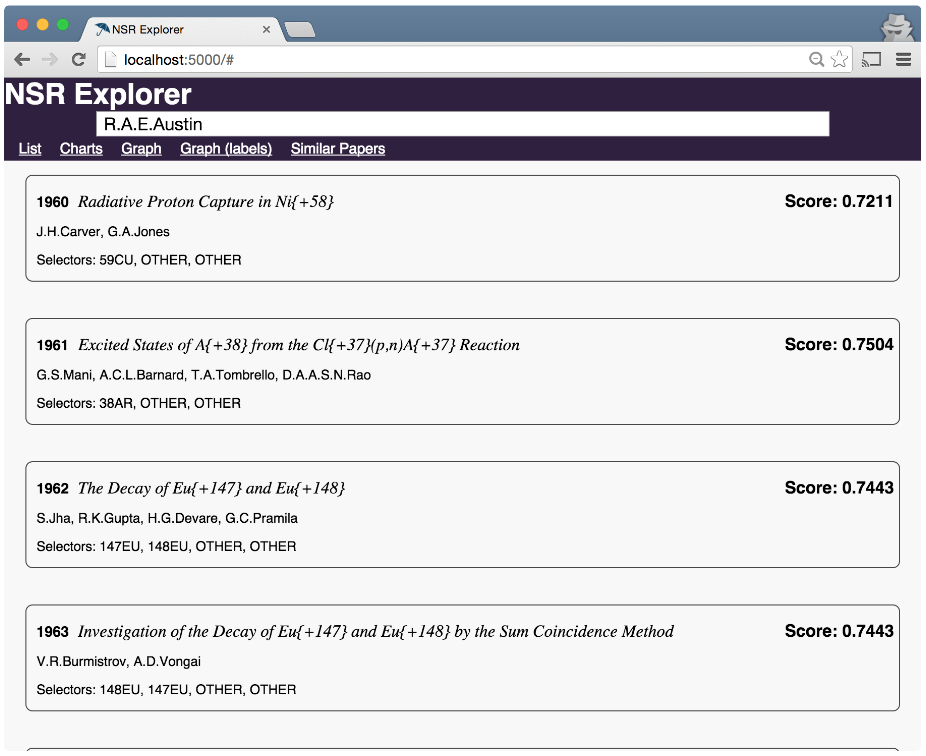


Figure 6: The graph-labels view for "R.A.E.Austin"



1960 *Radiative Proton Capture in Ni{+58}* **Score: 0.7211**
J.H.Carver, G.A.Jones
Selectors: 59CU, OTHER, OTHER

1961 *Excited States of A{+38} from the Cl{+37}(p,n)A{+37} Reaction* **Score: 0.7504**
G.S.Mani, A.C.L.Barnard, T.A.Tombrello, D.A.A.S.N.Rao
Selectors: 38AR, OTHER, OTHER

1962 *The Decay of Eu{+147} and Eu{+148}* **Score: 0.7443**
S.Jha, R.K.Gupta, H.G.Devare, G.C.Pramila
Selectors: 147EU, 148EU, OTHER, OTHER

1963 *Investigation of the Decay of Eu{+147} and Eu{+148} by the Sum Coincidence Method* **Score: 0.7443**
V.R.Burmistrov, A.D.Vongai
Selectors: 148EU, 147EU, OTHER, OTHER

Figure 7: The similar papers view for “R.A.E.Austin”

2 The Data and Database

The United States National Nuclear Data Center (NNDC) has composed the Nuclear Science References (NSR) database. A full database dump of the NSR was acquired on January 29th 2014 [4]. For simplicity, the data acquired from the NNDC on that date will be referred to as if it were the complete NSR database. All efforts have been taken to ensure the research procedures can easily be extended and repeated on new NSR data.

The work discussed in this chapter is motivated by the need to easily retrieve information from the NSR data and manipulate it to facilitate answering questions. We first discuss the NSR data as provided by the NNDC [4]. Special attention is given to the keyword abstracts as the metadata they provided is what sets the NSR data apart from other bibliographic databases. Then the method for converting the provided raw data into our JSON representation is discussed. The MongoDB database software is introduced and its aggregation framework is discussed. The mechanics of transform and importing the data are not fully described in this text. Instead, the conversion and importing procedures have all been scripted and made available as included source code in the Appendix.

2.1 NSR Data

The NSR has 9 possible types of fields which are shown in Table 1. Each entry can only have one of each field type except for `<KEYWORDS>` and `<SELECTRS>` which exist as a pair and an entry can have multiple pairs of them. An example of the raw data for a single paper can be seen in Snippet 1.

```
<KEYNO   >1988AB01                                                             &
<HISTORY >A19880309 M19880315                                                  &
<CODEN   >JOUR PRVCA 37 401                                                    &
<REFRENCE>Phys.Rev. C37, 401 (1988)                                            &
<AUTHORS >A.Abzouzi, M.S.Antony                                                &
<TITLE   >Calculation of Energy Levels of {+232}Th,{+232}{+-}{+238}U for K(|p) =&
 0{++} Ground State Bands                                                      &
<KEYWORDS>NUCLEAR STRUCTURE {+232}Th,{+232},{+234},{+236},{+238}U; calculated le&
vels,band features. Semi-empirical formalism.                                  &
<SELECTRS>N:232TH;A. N:232U;A. N:234U;A. N:236U;A. N:238U;A. C:OTHER;A.        &
<DOI     >10.1103/PhysRevC.37.401                                              &
```

Snippet 1: An example NSR entry showing the raw NSR EXCHANGE data format.

Table 1: The nine legal record identifiers from the Nuclear Science References Coding Manual [5].

Identifiers	Description
`<KEYNO   >`	Reference keynumber
`<HISTORY >`	Administrative record
`<CODEN   >`	Standard form reference
`<REFRENCE>`	Free text reference
`<AUTHORS >`	Author names
`<TITLE   >`	Reference title
`<KEYWORDS>`	Keyword abstract
`<SELECTRS>`	Indexing parameter list
`<DOI     >`	Digital object identifier

The `<KEYNO   >` field is a unique key number assigned to each NSR entry. The date on which a particular entry was added to the database or last modified is encoded in the `<HISTORY >` field. The `<CODEN   >` and `<REFRENCE>` fields contain information about the journal or other type of resource the document came from. The `<AUTHORS >` field is a comma-separated list of author names. The author list is one of the key relational components of the data, establishing links between NSR entries and other authors. The `<TITLE >` field is a free text field representing the title of the reference with a custom set of abbreviations for special characters like Greek letters. These abbreviations are detailed in the NSR coding manual [5]. In this work, the abbreviations have been translated to LaTeX. The `<DOI >` field contains the digital object identifier code that uniquely links to the source document's metadata. While not strictly necessary, the DOI often has a URL associated with it that links to the source document on the website of the publishing journal [6]. The two fields, `<KEYWORDS>` and `<SELECTRS>` have the most structure and require special attention which is given in Keyword Abstracts.

2.1.1 Keyword Abstracts

The `<KEYWORDS>` field , or keyword abstract, is written by the maintainers of the NSR database, and then used to generate the `<SELECTRS>` field [5]. Each NSR entry is read and then a keyword abstract is manually created to reflect the physical systems that were studied and measured in the work.

> "What distinguishes NSR from more general bibliographic databases is the level of detail provided in the keyword abstracts." [5]

Keyword abstracts each have one of the following major topics: `NUCLEAR REACTIONS`, `RADIOACTIVITY`, `NUCLEAR STRUCTURE`, `NUCLEAR MOMENTS`, `ATOMIC PHYSICS`, `ATOMIC MASSES`, and `COMPILATION`. To accommodate work that spans multiple topics, NSR entries can have multiple keyword abstracts. Following these major topics are one or more indexed sentences. These sentences describe the elements of the physical system studied, and any measurements that were made. For example, the `<KEYWORDS>` field in Snippet 1 encodes that the referenced work calculated energy (implied) levels and band features using a semi-empirical formalism for ^{232}Th, ^{232}U, ^{234}U, ^{236}U, and ^{238}U.

It is this structure that provides the most semantic information about the NSR entry. Thanks to the careful work of the NSR maintainers, the `<KEYWORDS>` and resulting `<SELECTRS>` fields reveal the NSR entry's content in a machine-readable manner. Without this information any data mining project using the content of the NSR entries would require raw text access to the either the full document or the abstract. Getting full text access to thousands of papers is often significantly challenged by copyright laws.

The selectors are computer generated from the keyword abstracts. The schema used in this work, as discussed in Data Representation, has `<SELECTRS>` parsed into a 3 dimensional array with `type`, `value`, and `subkey` variables. The following quote from the NSR Coding Manual [5] describes the valid `type`s:

> N, T, P, G, R, S, M, D, C, X, A, or Z, which stand for nuclide, target, parent, daughter, reaction, subject, measured, deduced, calculated, other subject, mass range, and charge range, respectively. [5]

The type of data for `value` changes based on the value of the `type`. For `types` N, T, P, and G, the `value` is a nuclide written in the form AX with A equal to the mass number, and X equal to the chemical symbol. The value for A may have any number of digits. X may be one, two, or three letters. The `subkey` variable is used to link together multiple selectors of the same keyword sentence.

2.2 Data Preparation

The NSR data are maintained in a custom EXCHANGE format [5]. This format is flat text that is not suitable for direct analysis. The data needs to be parsed into data structures for analysis and use. The approach least likely to introduce errors is to transform the data into a common format for which parsers already exist.

JavaScript Object Notation, or JSON, was chosen as the data format for this work. While other data formats could have sufficed (perhaps YAML, for example), certain common data formats like comma-separated values (csv) would have been more difficult. JSON met the following requirements: support for arrays, openly available, well-supported, with an established user community, and it was familiar to the author. The requirement for array support is discussed further in the Data Representation section.

Each NSR entry will be represented as a JSON object.[1] JSON objects are composed of keys and values. A key is a unique string that maps to a value. A value can be a string, a number, an array, or another object. Similarly, arrays can contain strings, numbers, objects or additional arrays. The arrays in JSON may be considered as ordered lists by some as they can contain elements of mixed types. However, the arrays in the NSR data do not contain mixed types. Snippet 2 shows an example of a JSON object and the final representation of an NSR entry.

Transforming the NSR data to a collection of JSON objects is possible with a series of search and replace commands using regular expressions. The commands are recorded in the Perl[2] script `parseNSRtoJSON.pl` available at github.com/valencik/mastersAPSC. The result of the scripts is a file with a valid JSON object for each NSR entry. The scripts can be used to reproduce the transformation data as new NSR data becomes available.

The data representation is the result of careful consideration of the types of queries to be made on the data. The data schema uses data types that best reflect how the data will be used. This is important as the data schema will

[1] These objects are referred to as documents once stored in the database. See The Database.

[2] Perl is used here as it remains one of the best RegEx implementations, and allowed for scripts that read as a simple ordered list of transformations to apply.

determine the types of queries we can make on the database. For example, with data spanning 120 years, it is helpful to filter the data based on a numeric year value. As such the `year` value in the data schema is an integer. This allows the construction simple queries to get NSR entries from a specific year or from a year range. Queries and example code are discussed in The Database.

The list of authors for a NSR entry is a more complicated data type as it involves multiple elements. It is best represented as an array of strings, with each unique author being a separate string element in the array. The representation of the author list as an array instead of a free text field is beneficial as the author list is now a data structure. With this structure comes information and ease of computing different properties of that data. The length of the array tells us how many authors collaborated on a given NSR entry. And since arrays are ordered, we can easily determine the first author[3] of an entry. While it is possible to extract the same information from a free text field, parsing our data into data structures creates structures that are compatible with many tools, such as MongoDB. Users of the database can now sort entries by their number of authors, or count the number of times someone was first author. Additionally, almost every aggregation query[4] made in our work relied on using array specific operations on the author array at some stage.

It is possible to store the NSR data in a relational model. In a relational database the authors would have their own tables, separate from papers, as they are separate entities. This inefficient choice would entail a table and data schema created for the papers and then a separate table and schema for the authors, and similarly for keywords, selectors, and history. It is more efficient to convert the original data into a data schema that uses arrays. This is the primary motivator for not using a standard relational database.

[3]The significance, if any, of being first author changes amongst journals. A clever data scientist will want to consider the `<REFRENCE>` information along with any first author analysis.

[4]See the MongoDB Aggregation Framework section for more details.

2.3 Data Representation

An example of the final data representation we use is shown in Snippet 2. It is a JSON object for the NSR entry with keynumber `1988AB01`. The `_id` value is a string used as the unique identifier in the MongoDB collection (as discussed in The Database), its value is the same as the `<KEYNO     >` of the original NSR entry. The `year` value is an integer and represents the year the resource was published. The `history` value is an array that contains encoded information representing dates when the original NSR document was added and/or modified. The `code` value is a string copy of the `CODEN` value in the NSR data. The `type` value is a string that describes what publication type (journal, thesis, conference paper, etc) the resource is. The `reference` value is a string copy of `REFRENCE` from the NSR data. The `authors` value is an array of string elements representing the authors who published that resource. The `title` value is a string, formatted for LaTeX, that represents the title of the resource. The `keywords` value is an array of strings that represent the KEYWORD sentences as described in the NSR manual. The `selectors` value is an array of objects that contain the type, value, and subkey information generated by the keyword entry by the NSR. The `DOI` value is a string of the Digital Object Identifier for the published resource. Finally, the `simPapers` value is an array that contains objects, where each object refers to another NSR entry that is above a minimum similarity threshold. The objects in `simPapers` are determined via calculation which is discussed in Similar Papers.

```
{
  "_id": "1988AB01",
  "year": 1988,
  "history": [ "A19880309", "M19880315" ],
  "code": "JOUR PRVCA 37 401",
  "type": "JOUR",
  "reference": "Phys.Rev. C37, 401 (1988)",
  "authors": [ "A.Abzouzi", "M.S.Antony" ],
  "title": "Calculation of Energy Levels of {+232}Th,{+232}{+-}{+238}U for K(\\pi
↪   ) = 0{++} Ground State Bands",
  "keywords": [ "NUCLEAR STRUCTURE {+232}Th,{+232},{+234},{+236},{+238}U;
↪   calculated levels,band features. Semi-empirical formalism." ],
  "selectors": [
    { "type": "N", "value": "232TH", "subkey": "A" },
    { "value": "232U", "subkey": "A", "type": "N" },
    { "type": "N", "value": "234U", "subkey": "A" },
    { "subkey": "A", "type": "N", "value": "236U" },
    { "type": "N", "value": "238U", "subkey": "A" },
    { "type": "C", "value": "OTHER", "subkey": "A" }
  ],
  "DOI": "10.1103\/PhysRevC.37.401",
  "simPapers": [
    { "score": 0.90890002250671, "paper": "1992BAZJ" },
    { "score": 0.89768290519714, "paper": "1994CH14" },
    { "score": 0.88365876674652, "paper": "1979FAZX" },
    { "score": 0.81536161899567, "paper": "1983DU10" },
    { "score": 0.8083301782608, "paper": "1979CH02" },
    { "score": 0.7871305346489, "paper": "1984PE01" },
    { "score": 0.76397824287415, "paper": "1960DU10" },
    { "score": 0.76397824287415, "paper": "1981SE07" },
    { "score": 0.76397824287415, "paper": "1995KU31" },
    { "score": 0.76397824287415, "paper": "1999BU03" },
    { "score": 0.76397824287415, "paper": "2011NA24" },
    { "score": 0.76335608959198, "paper": "1985ZH08" },
    { "score": 0.74211376905441, "paper": "1975IVZM" },
    { "score": 0.73295497894287, "paper": "1981MA35" },
    { "score": 0.73295497894287, "paper": "1996ZH29" },
    { "score": 0.72842478752136, "paper": "1979ES06" },
    { "score": 0.72127419710159, "paper": "1982MI12" },
    { "score": 0.72127419710159, "paper": "1983MI19" },
    { "score": 0.70403093099594, "paper": "1973IM02" }
  ]
}
```

Snippet 2: An example NSR entry showing the final data representation as a JSON object. Note that newlines are ignored in JSON and only present for readability. Also note that the order of keys and values in an object is not garunteed to be preserved, as can be seen from the `selectors` objects.

2.4 The Database

Database systems are an important tool in information retrieval. Large datasets should be organized in databases to provide useful abstractions for users. E. F. Codd discusses this in his 1970 paper [7], as such the idea is not new nor uncommon. In the Data Preparation and Data Representation sections we discussed the transformation of the provided data into our own representation. The users of our application, NSR Explorer, should not be concerned with the internal representation of data.[5] The resulting data representation is imported and stored in a database to enable the NSR Explorer web application to query the data. This section will detail our choice in database software, how queries are made, and briefly introduce some performance considerations.

2.4.1 MongoDB

MongoDB is an open source NoSQL document store database system. It was chosen because it is open source, easy to use, well supported, and the author is familiar with it. Additionally it has nice features such as JSON support, an aggregation framework (see MongoDB Aggregation Framework), and is easy to set up.

Other NoSQL databases like CouchDB support JSON and may have been acceptable as well. MongoDB and CouchDB are both comparatively new database systems. Postgres also supports JSON and is a mature database system. Despite the prevalence of MySQL, it was not chosen because it is a relational database and would thus not support the arrays in the data schema as outlined in Data Preparation.

In a relational database system such as MySQL, each NSR entry would have to

[5]The work in [7] goes further to suggest that even the developers of the NSR Explorer application should not be concerned about the internal data representation. This should be abstracted by the relational model of data discussed in that work.

be split up, with different pieces of information populating different database tables. Authors would be a type of entity in their own authors table, that each NSR entry in an NSR table would link to. This type of relationship would be necessary for keywords and selectors as well.

As reported in the section Data Preparation, a JSON object was constructed for each entry in the NSR data. We create a database in MongoDB titled `masters`. This database will hold multiple collections. MongoDB collections store multiple documents. Each JSON object is a document in MongoDB terminology. To populate the MongoDB database, these JSON structures were flattened into a single file, and imported into a MongoDB collection using the `mongoimport` tool. After importing was completed, there were 212835 documents in the MongoDB collection, one for each entry in the NSR database.

The `pymongo` module [8] can be used to query our database using the Python programming language. Example python code to retrieve all NSR entries from the 1970s is given in Snippet 3. The first line imports the `pymongo` module which enables communication with a MongoDB database. We then establish a connection with the local database titled `masters` and save that connection in the `db` object. Finally we use the `NSR` collection of the database and pass the `find` method our query expressed as a JSON object. Note the use of `$gte` and `$lt` which correspond to the mathematical operators greater than or equal to and less than.

```
import pymongo
db = pymongo.MongoClient()['masters']
db.NSR.find({"year": {"$gte": 1970, "$lt": 1980}})
```

Snippet 3: Python code to get all NSR entries from 1970 to 1979.

2.4.2 Indexing the Data

The performance of the database can be optimized by indexing on important or frequently referenced fields such as "authors" and "year". Indexing speeds up search queries in a manner similar to sorting a series of data elements. MongoDB allows for many different types of indexes. We create single field indexes on the `_id`, `year`, `authors`, `selectors.type`, `selectors.value`, and `type` fields[6]. This enables fast lookups for documents[7] according to the indexed fields. For example it would be quick to find all the documents with type 'Journal' and year '1983'. Text indexes can also be created to enable fast search of words in the titles or keyword fields. This has not been done as the titles and keyword fields are not used in our analysis. We instead make use of the list of selectors to infer the topic of a given NSR entry.

There are additional concerns in hosting a database server and web application. Typically a database is hosted on a dedicated server, separate from the web application, and perhaps not publicly facing. These issues, and additional performance configurations will not be further addressed in this work. They are however addressed in the code repository for this work available at: github.com/valencik/mastersAPSC.

2.4.3 MongoDB Aggregation Framework

The MongoDB aggregation framework is powerful and enables data manipulation similar to that obtain in SQL[8] via the `GROUP BY` operation [9]. There are a handful of simple aggregation operations that can be piped together to build

[6]We use dot notation to denote that `selectors.type` refers to the `type` field of the `selectors` object.

[7]Recall that MongoDB is a 'document' store database, and each NSR entry has been imported as a 'document' in the MongoDB collection.

[8]SQL or Structured Query Language is a common programming language for interacting with data.

complex queries. All aggregation operations take in a list of data documents, manipulate them in some way, and then output the results to the next operation.

The `match` operation acts as a filter, returning only the documents that meet the specified criteria. The `project` operation manipulates each individual document renaming, omitting, or changing each field according to the input parameters. The `unwind` operation acts on an array field of the input documents. It creates a new document for each element in the array, with all fields duplicated except the array field which is equal to the element. The `group` operation can combine similar documents and can perform calculations based on that combination. A common usage is to sum a value, perhaps price, of all the input documents.

There are some additional, more straightforward, operations such as `sort`, `limit`, `skip`, and `redact`. The final results from an aggregation query can be saved to a collection using the `out` operation, or can be returned to the calling application through the many MongoDB APIs.

MongoDB is currently a popular database and there exist tutorials and example applications. The MongoDB documentation is well written and provides a good overview of the aggregation framework [10]. All MongoDB interactions in this work use the python driver, `pymongo` [8].

2.4.4 Future Work - The Database

An extension to this work is to support additional database systems. The prevalence of MySQL is motivation to support it. However, in continuing with the desire to use a NoSQL database system, the work could be extended to support CouchDB with relative ease.

2.5 Conclusion

We have described the NSR data as provided by the NNDC [4]. Scripts have been prepared to transform the NSR EXCHANGE format into a list of JSON objects, one for each NSR entry. The JSON objects have been imported into a MongoDB database called `masters` in a collection called `NSR`. Basic queries as well as aggregation queries have been introduced. The system architecture for querying the data is complete as described. The sections following this will describe different analyses and results from querying the data.

3 Data Summarization

This chapter introduces the queries that can be run on the database constructed in The Data and Database section. Examples are shown that filter the data using constraints on different data types such as year ranges or author names. Bar charts and pie charts are introduced for visualizing the data.

Through data summarization we can reveal first-order characteristics of the data set. Our goal was to make available a broad perspective of its structure and composition.

3.1 Data Composition and Queries

We can construct queries to reveal the composition of the NSR data. For example, there are 212835 entries[9] that span from 1896 to 2014. We can answer questions such as "What percentage of all entries are journal articles?" As Table 2 shows, the majority of the document types in the NSR are journal articles. The next most popular are reports and conference proceedings. There are fewer books and preprints than there are unknown and unlabeled entries. The python code to produce these results is shown in Snippet 4.

Table 2: The amounts of each type of NSR entry in the whole data set.

Type	Amount	Percentage
THESIS	1934	0.908%
PREPRINT	779	0.366%
BOOK	107	0.050%
PC	1661	0.780%

[9] Recall that the data set used in this work is a snapshot of the entire NSR data as downloaded in January 2014.

Type	Amount	Percentage
CONF	16836	7.910%
REPT	24554	11.53%
JOUR	165477	77.74%
UNKNOWN	1487	0.698%

```python
import pymongo
db = pymongo.MongoClient()['masters']
db.NSR.aggregate([{"$group": {"_id": "$type", "count": {"$sum": 1}}}])
```

Snippet 4: The aggregation query to get amount of types in the NSR.

The summarization analysis can conveniently be applied to subsets of the data. The data can be filtered to only involve a particular author. This provides answers to questions such as "what percentage of A.J.Sarty's contributions were journal articles?" Table 3 shows A.J.Sarty has primarily worked on journal articles, with one preprint article. The code for this query, which is shown in Snippet 5, simply adds a `$match` operation to Snippet 4.

Table 3: Different types of NSR entries for author A.J.Sarty.

Type	Amount	Percentage
PREPRINT	1	5%
JOUR	21	95%

```python
import pymongo
db = pymongo.MongoClient()['masters']
db.NSR.aggregate([{"$match": {"authors": "A.J.Sarty"}},
                  {"$group": {"_id": "$type", "count": {"$sum": 1}}}])
```

Snippet 5: Aggregation query to get the types of an author's publications.

The data can also be partitioned or sliced in time, supporting questions such as “what percentage of 1989 entries are journal articles?” As we can see from Table 4 72.06% of the NSR entries in 1989 are journal articles. This percentage is different than that of the whole work (as shown in Table 2), but not by a significant amount. It does highlight a particular point of interest, i.e. that the data are not uniform.

Table 4: Different types of NSR entries in 1989.

Type	Amount	Percentage
THESIS	16	0.398%
PREPRINT	21	0.523%
BOOK	7	0.174%
PC	14	0.348%
CONF	331	8.248%
REPT	732	18.24%
JOUR	2892	72.06%

For a particular selection of NSR data, it is useful to know the rankings for important data fields. For example, when a user searches an author on the application they are presented with a ranked list of their most frequent coauthors, keywords, and nuclides. This type of analysis can of course be applied to the whole dataset as well. Table 5 shows the authors with the highest count of NSR entries in the entire database.

Table 5: The top 10 most prolific authors in the NSR database.

Author	Number of Publications
R.V.F.Janssens	992
M.P.Carpenter	787

Author	Number of Publications
A.Faessler	736
J.H.Hamilton	703
I.Ahmad	694
B.A.Brown	690
I.Y.Lee	671
W.Greiner	637
A.O.Macchiavelli	624
T.L.Khoo	614

3.2 Author Contributions

There is a wide range in publication numbers among the roughly $100,000$ authors in the NSR. Table 5 shows the upper bound in publication numbers, and 41254 authors share the lower bound of one publication. These are authors with different publication traits (a topic that is further explored in Cluster Analysis). We want to enable discovery about the structure of author contributions. Are the majority of NSR entries contributed by the many authors who publish once or the few with hundreds of publications?

The database can be used to answer a similar question: How many of the NSR entries are affected if every author who contributed to fewer than a given number of entries is removed? First, every paper is taken and duplicated for every single author in that paper's author list. There is now a database object for each author in each paper. Each time an author appears their publication count increments. Next, each database object that has an author with a publication count below the cutoff is removed. Finally, the unique remaining NSR entries are the ones that have authors with more than the given publication count. Table 6 shows the number of NSR entries that remain after all the authors with a specified publication count are removed[10]. Note that the starting number is 190654 not 212835 as quite a few NSR entries do not have an author field. Table 7 shows a breakdown of the NSR entry types that do not have authors. A large percentage of those entries without author fields are reports and conference proceedings[11].

Table 6: NSR entries affected by removal of authors with a publication count less than the cutoff.

Entry Number Cutoff	Entries Remaining	Difference
1	190654	
2	187741	2913
3	185404	2337
4	183410	1994
5	181315	2095
6	179606	1709
7	177945	1661
8	176390	1555
9	174702	1688
10	173117	1585
11	171509	1608

Table 7: Types without any authors.

Type	Amount
UNKNOWN	33
PC	48
CONF	4614

[10]The code to produce the results in Table 6 is shown in Appendix Snippet 12

[11]Of the 5564 journal articles without an author field, 5489 were written between 1970 and 1980.

Type	Amount
PREPRINT	78
THESIS	968
REPT	10876
JOUR	5564
Total	22181

The relationship between publication amount cutoff and NSR entries remaining is consistent. The same data in Table 6 is plotted in Figure 8.

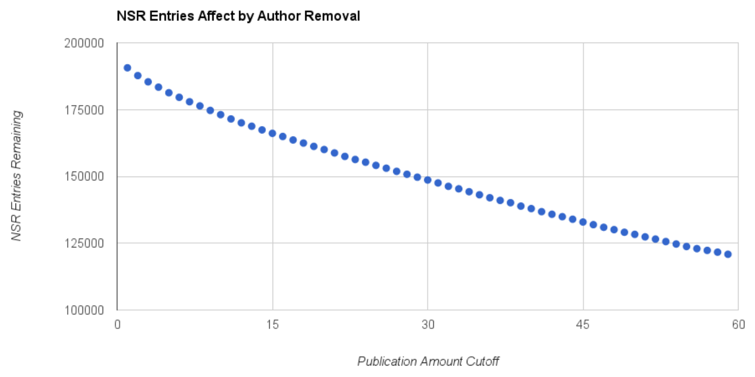


Figure 8: NSR entries remaining after author removal

The values presented in Table 6 suggest that the bulk of the entries in the NSR are associated with authors who publish more than just a few times. Taking the last value in the table, the authors who publish 11 or more times in the NSR make up about 90% of all the NSR entries with an author field. There are only 18006 authors who have published 11 or more times. Therefore, about 18% of authors make up about 90% of the contributions in the NSR database.

3.3 Visualizations

Visualizations provide a summary of data at a glance. Consider Figure 9, which quickly demonstrates that the majority of NSR entries were published in the last 50 years.

The fact that there were comparatively low publication numbers in the first 50 years was a useful property of the dataset. It permitted testing data analysis code on small portions of the data (years pre 1950), before applying the code to the full dataset. This was helpful in developing the network analysis code and visualizations, as post 1950 the networks are too large to process quickly.

There are two primary visual methods for displaying summary information in this application: histograms and pie charts. The histograms, as seen in Figure 9, can show how a slice of the database evolves over time. They are also useful to see amounts in categorical data. Figure 10 shows the amount of each different document type in the NSR database.

The pie charts demonstrate the relative sizes of portions of the data. The document type amounts are shown in Figure 11 as a pie chart. Figures 10 and 11 are visual representations of the data in Table 2.

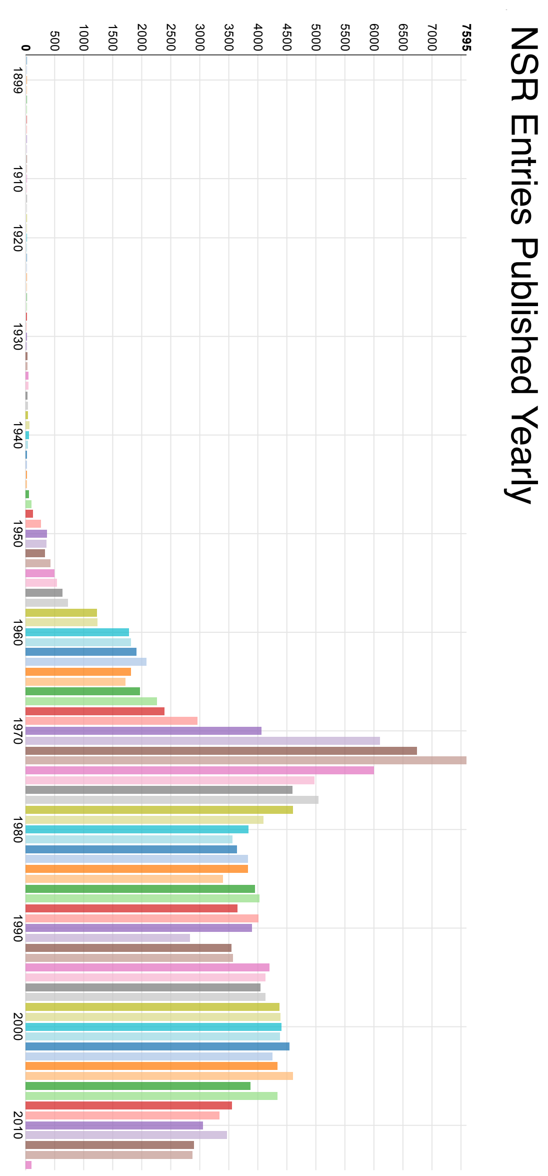


Figure 9: A histogram of all NSR entries published from 1896 to 2014

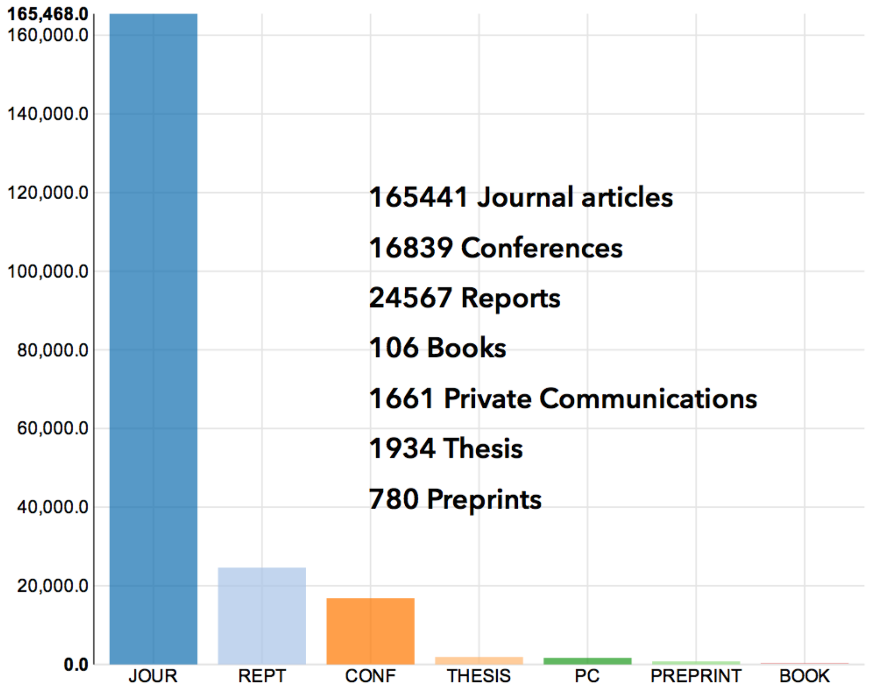


Figure 10: A histogram showing the contribution types over all years

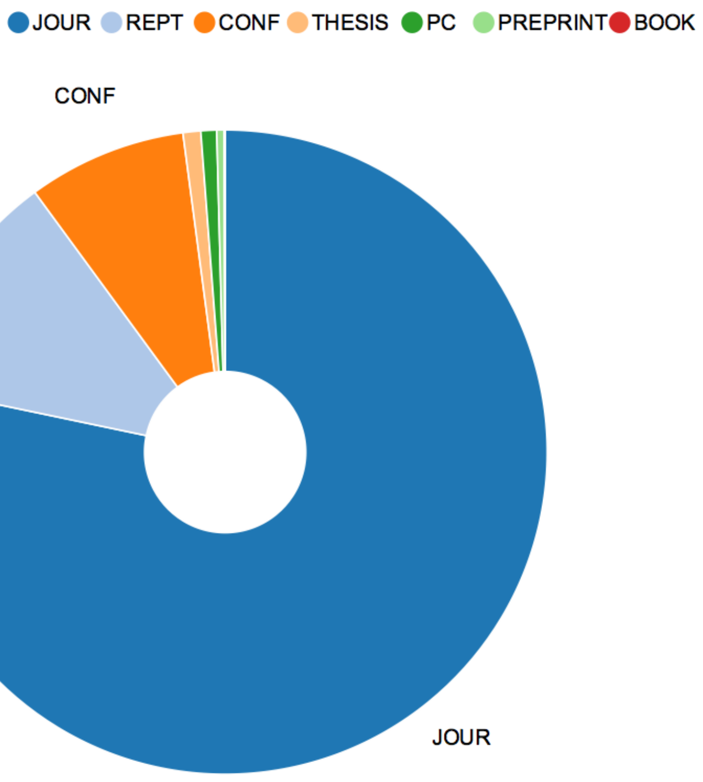


Figure 11: A pie chart showing the types of NSR entries over all years

4 Network Analysis and Visualization

This chapter explores the analysis of the NSR data as a network graph. The list of authors of a paper is used to build a network graph of authors and their copublication relationships. In this work, the word ‘graph’ will always refer to the mathematical representation of a set of objects and their links.

Enabling exploration of the NSR data has been a core motivation for this work. Transforming the NSR data into a network graph has made new questions and analyses accessible. Additionally, network graphs lend themselves well to interesting visualizations which has been another motivator.

This thesis does not consider the analysis of specific network graphs but rather enables the NSR Explorer users to easily do so.

4.1 Data Graphs

The first graphs constructed in this work had each node represent an author, and each edge or link represent a coauthorship. An example can be seen in Figure 12.

Figure 12 is a single component of the complete 1940 author graph (shown in Appendix Figure 30). It has 7 nodes, each of which is a different author, and 12 edges, which represent a coauthorship between the two nodes. `M.Ikawa` has published with everyone in the graph. We can use this knowledge in a database query to get the entries that make up this graph (see Snippet 6).

```python
import pymongo
db = pymongo.MongoClient()['masters']
db.NSR.find({"year": 1940, "authors": "M.Ikawa"})
```

Snippet 6: Python code to get the NSR entries in Figure 12.

The results of the database query (shown in Snippet 7) reveal there were 3 entries

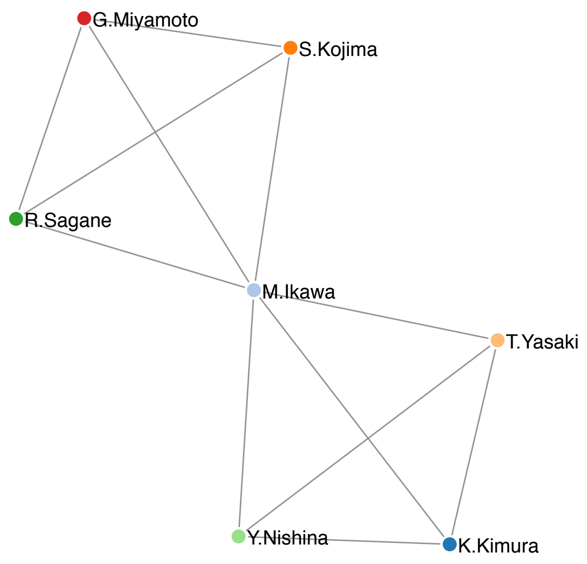


Figure 12: A single component of the 1940 author graph.

in 1940 that contributed to this graph. One paper titled “Fission Products of Uranium by Fast Neutrons” has authors `Y.Nishina`, `T.Yasaki`, `K.Kimura`, and `M.Ikawa`. The other entries, titled “Neutron Induced Radioactivity in Columbium” and “Artificial Radioactivity Induced in Zr and Mo” respectively are both authored by `R.Sagane`, `S.Kojima`, `G.Miyamoto`, and `M.Ikawa`. This demonstrates a limitation in the current graph visualization. `G.Miyamoto` and `M.Ikawa` have published together twice (in 1940) but their edge looks no different than the edge between `Y.Nishina` and `T.Yasski`. Most graph libraries allow for customization and embedding of data. In future work, the graphing routines could be modified to represent the number of copublications along the graph edges. This could be achieved with a text label, an added thickness to the edge line, or colour.

```
{
  "_id": "1940NI03",
  "year": 1940,
  "history": [
    "A19800701",
    "M19860317"
  ],
  "code": "JOUR PHRVA 58 660",
  "type": "JOUR",
  "reference": "Phys.Rev. 58, 660 (1940)",
  "authors": [
    "Y.Nishina",
    "T.Yasaki",
    "K.Kimura",
    "M.Ikawa"
  ],
  "title": "Fission Products of Uranium by Fast Neutrons",
  "DOI": "10.1103\/PhysRev.58.660"
}
{
  "_id": "1940SA06",
  "year": 1940,
  "history": [
    "A19800701",
    "M20010110"
  ],
  "code": "JOUR PPMJA 22 174",
  "type": "JOUR",
  "reference": "Proc.Phys.-Math.Soc.Japan 22, 174 (1940)",
  "authors": [
    "R.Sagane",
    "S.Kojima",
    "G.Miyamoto",
    "M.Ikawa"
  ],
  "title": "Neutron Induced Radioactivity in Columbium"
}
{
  "_id": "1940SA08",
  "year": 1940,
  "history": [
    "A19800701",
    "M19980402"
  ],
  "code": "JOUR PHRVA 57 1179",
  "type": "JOUR",
  "reference": "Phys.Rev. 57, 1179 (1940)",
  "authors": [
    "R.Sagane",
    "S.Kojima",
    "G.Miyamoto",
    "M.Ikawa"
  ],
  "title": "Artificial Radioactivity Induced in Zr and Mo",
  "DOI": "10.1103\/PhysRev.57.1179"
}
```

Snippet 7: JSON documents for the NSR entries in Figure 12.

We can also visualize graphs with multiple components. Disconnected components, like those visible in Figure 13, are groups of authors who have published together and not with any author in another component. There is only ever one node per author identifier.

Most graphs produced from yearly data queries have multiple components. A graph produced by papers including a single author will have a single component by definition. As the number of NSR entries forming the graph gets larger, the main connected component gets much larger than the rest of the components. Researchers at Facebook have done some interesting work confirming this on the largest social network studied [11]. They calculated that 99.91% of the 721 million users considered were in a single connected component.[12]

The visualization of large graphs is computationally intensive and produces complex images. All the graphs produced in the web application use a modified version of Mike Bostock's Force Directed Graph.

Above a certain size, these images are of questionable usefulness. The resulting shape or 'layout' of a graph is dependent on the graph layout algorithm used. Figures 14 and 15 use the same input data and two different layout algorithms (Yifan Hu ML and Atlas 2 respectively). The position of the nodes and edges in these figures are products of the layout algorithm used. The two figures are visually different enough to suggest that their positional information is meaningless.

The colour of the nodes is determined by Gephi's modularity function [12]. The modularity of a graph is a measure of structure. The graph is partitioned into communities where there are dense intra-community connections and sparse inter-community connections.

[12]The work in [11] also serves as an accessible overview of the types of analysis one might want to do on a network graph.

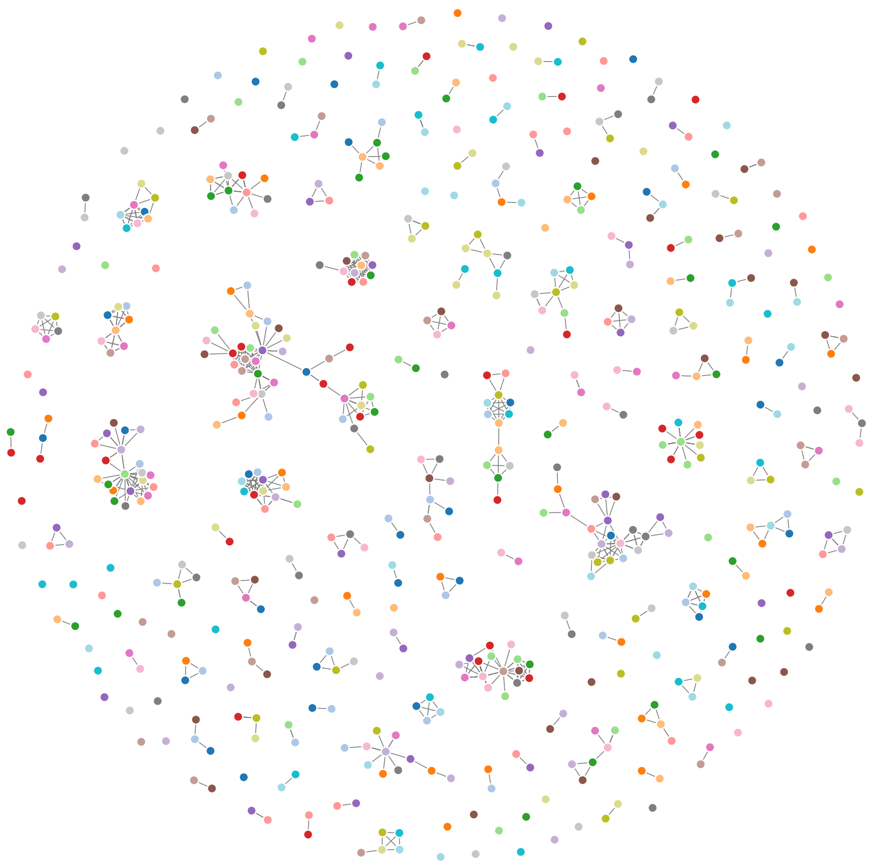


Figure 13: Network graph of the first 50 years of NSR data

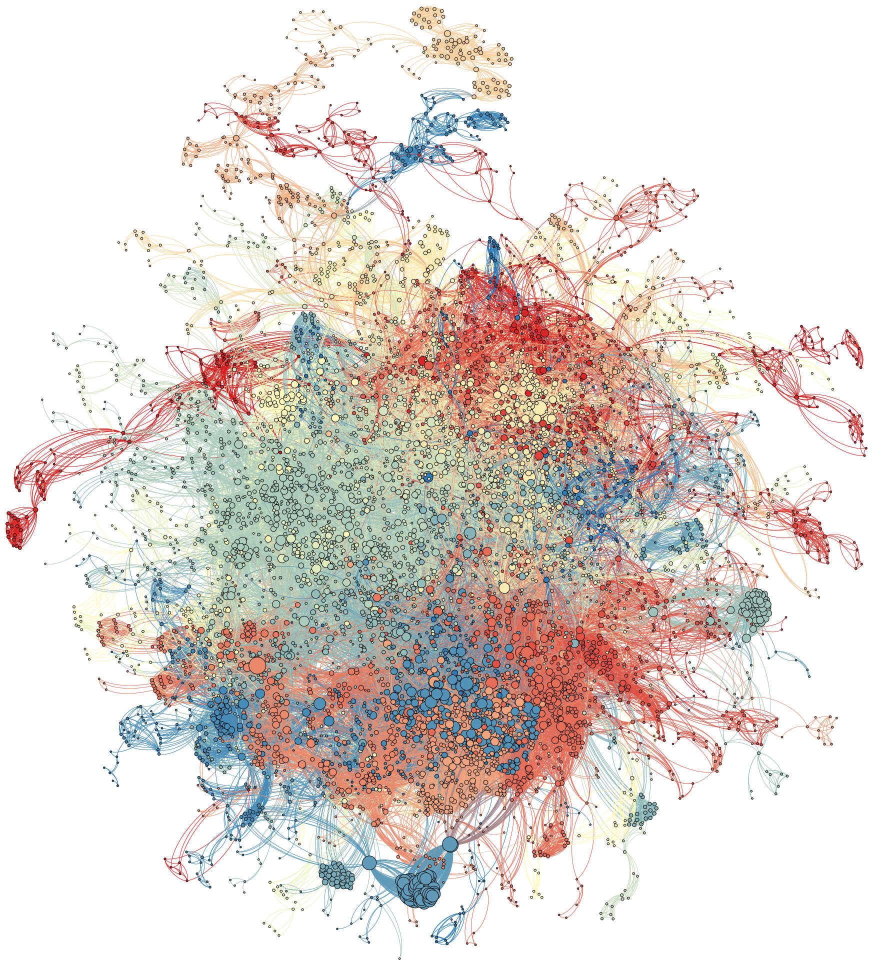


Figure 14: Network Graph of 1989 produced in Gephi with the Yifan Hu ML layout algorithm.

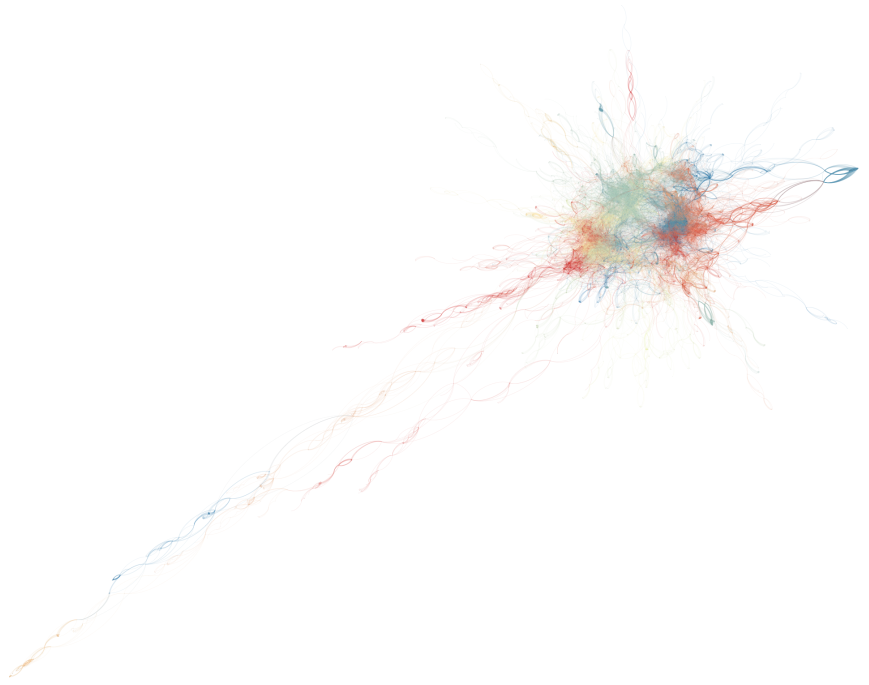


Figure 15: Network Graph of 1989 produced in Gephi with the Atlas 2 layout algorithm.

4.2 Nuclide Graphs

Almost any parameter can be used as a filter to produce an author network graph. The selector values present an interesting opportunity in this case. We can filter the NSR data to only include entries that involved a particular nuclide. Figure 16 shows an author node graph for all the NSR entries that have `LI11` as a selector value. The figure shows that there is one large connected component of the graph, and many smaller components. The largest connected component, alone, can be viewed by adding `topnetwork:1` to the input query.

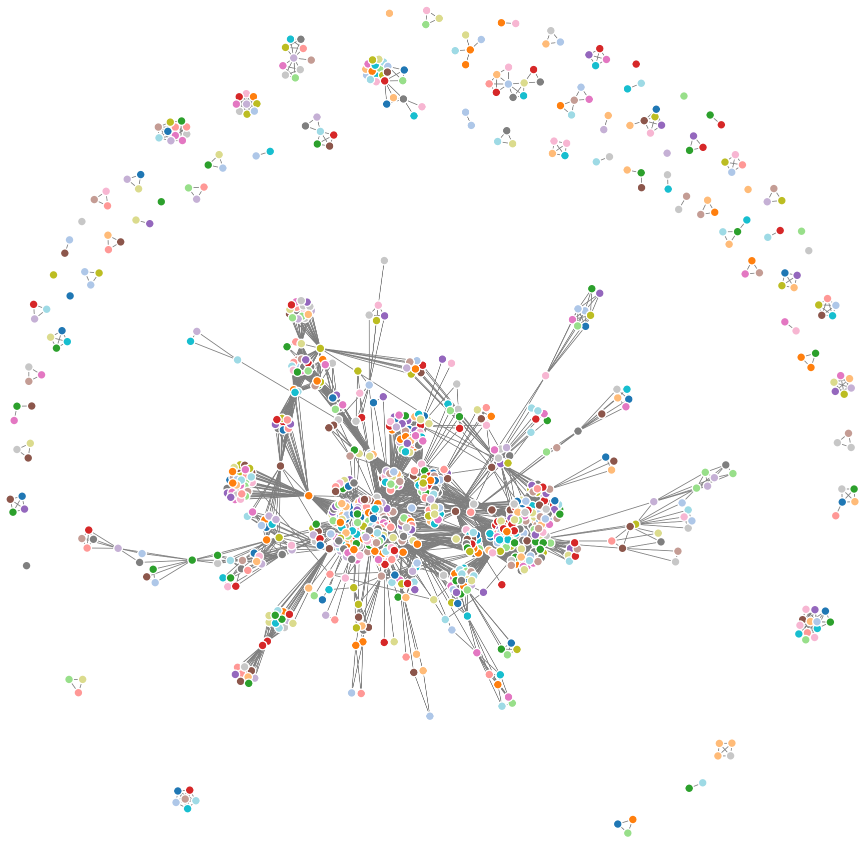


Figure 16: Network graph of authors publishing on Lithium-11

4.3 Implementation

The Python library Networkx was used to create the graph data structures, which can then be sent to our visualization code, or be exported for analysis with other tools. Networkx has a collection of algorithms and functions used to analyze and manipulate the graphs.

The Networkx library is primarily used for its graph data structure reading and writing methods. To be explicit, Networkx is used to create graphs from data returned by aggregation queries, and then convert the graph data into a format suitable for exporting to disk or to the NSR Explorer web application for visualization. The identification of connected components is the only algorithm provided by Networkx that is used in this work. For example, Figure 14 and Figure 15 use only the largest connected graph of all the NSR entries in the year 1989.

4.4 Exporting Graph Data

Treating the NSR database as graph data fulfils one of the primary goals of this thesis by opening up avenues for future work. All of the graphs we have created are constituted of authors as nodes with edges determined by their coauthors. These graphs are social networks of collaborating scientists. The study of them may be of interest to social scientists and network scientists.

Exporting the graph data offers efficiencies to future work. The Networkx library has support for writing the graph data structures to multiple file types, such as `gexf`, `GML`, `GraphML` and others. These files can then be imported into other analysis applications like Gephi. Example code for exporting the 1989 data to `gexf` format is available at github.com/valencik/mastersAPSC.

5 Text Mining

Text mining is an area of analysis that focuses on extracting useful information from unstructured plain text data. The data to analyze is often natural language text written by humans. Examples of such data include user reviews of a product or service, customer feedback comments, emails, forum posts, or even academic journal articles. Example goals of analyzing such works might be summarization, determining sentiment, finding topics, or finding similar items.

In this section, two distinct goals of this work were achieved using tools commonly employed in text mining. Recommending similar papers for a given selection of papers was executed using cosine simularity in a vector space model. Determining authors who may have multiple identifiers in the data was accomplished using string edit distances. Reducing instances of multiple identification of individuals in the NSR dataset facilitates later work by network analysts and social scientists.

A common task to both of these goals is comparing text. In order to compare two items we need a metric by which we can measure them. The comparison of individual strings as needed in the Author Name Analysis was done with string edit distances as discussed in that section. The comparison of documents or whole NSR entries requires another technique that operates on words instead of individual characters. The vector space model was ultimately chosen as mentioned above and fully discussed in the Vector Space Model section. However, a topic modelling method is discussed in the Future Work - Text Mining section of this chapter.

Introduction to Information Retrieval [13], or the Standford IR Book, serves as an excellent and freely available resource introducing concepts and techniques in information retrieval.

5.1 Vector Space Model

A substantial portion of the functionality of the NSR Explorer as an information retrieval system is provided by simple queries to the database. However, in developing the similar paper recommendation system, we make use of the vector space model,[13] an early information retrieval model [16] [13]. The vector space model is simple and powerful. Documents are represented as vectors where each dimension is a separate term. There are multiple ways of calculating the value for each dimension such as tf-idf or bag of words [17] [18]. The work done in the Similar Papers section uses 0 or 1 to represent whether a particular selector was present in the NSR entry.

As an example, let document one, $d1$, be “The quick brown fox jumps over the lazy dog” and document two, $d2$, be “The brown dog jumped over the brown fox”. We will model these two documents using a vector space model. A term vector for each document is created using a dimension for each separate term occuring in the collection of documents. The final vectorization of these two documents is shown in Table 8.

Often in information retrieval systems the most frequent words, like “the” or “a” are omitted. Consider the 5th and 4th of $d1$ and $d2$, “jumps” and “jumped”. These words are clearly similar, but are strictly different terms and would thus be represented with different dimensions after vectorization. Word stemming addresses this issue by trimming the suffix of words such that we only use the root word. The first stemming algorithm was published in 1968 [19] [20]. This algorithm was later improved by Porter in 1980 and has since become widely used thanks to the author making his continued improvements freely available [21] [22].

[13]The correct earliest citation for the vector space model used today is not trivially found. Frequently [14] is cited because of its title “A vector space model for automatic indexing”, however, as Dubin outlines in [15], this paper does not describe the vector space model as an information retrieval model.

Equations 1 and 2 are the term vectors for each document. The cosine similarity is the dot product of the two documents divided by the product of their magnitudes 3. [23]

Table 8: Term vectors for $d1$ and $d2$.

vector	quick	brown	fox	jump	over	lazy	dog
d1	1	1	1	1	1	1	1
d2	0	2	1	1	1	0	1

$$d1 = \langle 1, 1, 1, 1, 1, 1, 1 \rangle \tag{1}$$

$$d2 = \langle 0, 2, 1, 1, 1, 0, 1 \rangle \tag{2}$$

$$\cos(d1, d1) = \frac{d1 \cdot d2}{|d1|\,|d2|} \tag{3}$$

$$d1 \cdot d2 = (1)(0) + (1)(2) + (1)(1) + (1)(1) + (1)(1) + (1)(0) + (1)(1) = 6.0 \tag{4}$$

$$|d1| = \left((1)^2 + (1)^2 + (1)^2 + (1)^2 + (1)^2 + (1)^2 + (1)^2\right)^{\frac{1}{2}} = 2.645751311 \tag{5}$$

$$|d2| = \left((0)^2 + (2)^2 + (1)^2 + (1)^2 + (1)^2 + (0)^2 + (1)^2\right)^{\frac{1}{2}} = 2.828427125 \tag{6}$$

$$\cos(d1, d2) = \frac{d1 \cdot d2}{|d1|\,|d2|} = \frac{6.0}{|2.645751311|\,|2.828427125|} = 0.8017837257 \qquad (7)$$

As Equation 7 shows, the two documents are quite similar, and thus have a high cosine similarity. When $d1 = d2$ the similarity is 1.0. This technique is a simple way of numericizing text for further mathematical manipulation and treatment.

5.1.1 Similar Papers

A script was prepared to perform cosine similarity analysis on the NSR selectors. The code is available at github.com/valencik/mastersAPSC. For each NSR entry, a vector was formed from the entry's selectors. These vectors were used to form a corpus that calculated the frequency of each term in the vectors. Any selector with a value equal to `OTHER` was filtered out. These selectors are similar to stop words in text mining natural language data. Stop words[14], and these selectors, are the most commonly occuring, they contribute little meaning, and are therefore removed.

The vectors are formed by turning the selectors into strings. We dropped the `subkey` value for this analysis as we were not concerned with the ordering of the selectors. The fact that cosine similarity does not take into account the ordering of words is a limitation that negatively impacts its performance on real world text documents [24]. This does not affect our analysis as we were not analyzing natural language but constructing our "words" out of a list of items that act like keywords.

The python package `gensim` was used to handle the vector creation and similarity analysis. While `gensim` offers many features and different forms of similarity

[14]Common stop words in english are "the", "and", "it", "is", and "a".

measures[15], we made use of the cosine similarity routines.

The result was a list of paper `_id`s that were similar to the input paper in regards to their selector vectors. These results were written to the database in a new field `simPapers`. The `simPapers` field is actually an array of objects, similar to the `selectors` field. Each object contains two items, the `_id` of the paper, and the computed score from `gensim`. The usage of these similarity scores is shown in the Application Section.

5.2 Author Name Analysis

In this section, we describe the analysis which identifies and makes steps towards mitigating the issue of having multiple variants of author names in the database. We will use the word "author" (formatted plainly and without quotes) to refer to an individual human being who contributed to a work that is documented in the NSR database. The term "identifier" (also formatted plainly and without quotes) will refer to the string of text that occurs in the database. The actual identifier strings will always appear in a fixed width typeface. For example, an author may be Andrew Valencik, and he may have more than one identifier such as `A.Valencik`, `A. Valencik`, and or `A.C.Valencik`.

This analysis locates multiple identifiers in the database that may correspond to single authors. This may be caused by differences in style from one publication to the next, changes in formatting, or simple typos. Authors themselves may opt in some publications to be identified by more than one initial and only one in others.

Pritychenko reports 96200 unique authors in his 2014 paper [25]. However, at the end of the data preparation stage in this work, the database reported 100147 unique identifiers. An accurate total author count is not particularly important

[15]Documentation for `gensim`'s methods of calculating similarities is available at: radimrehurek.com/gensim/similarities/docsim.html

for this work creating a database and exploration application. However, correctly identifying and including all authors when doing network analysis *is* important.

There are 41254 unique identifiers that appear only once in the NSR database. Some portion of those are author name variances that only occur once. Knowing that portion is important to understanding something about the database; in the Initial Author Clustering section we investigated how the database changed as we removed identifiers (referred to as "authors" in that section) below a publication threshold. That analysis depended on correctly identifying the number of authors who had published a given number of times. Variances in the list of authors render such an analysis in inaccurate.

After the data preparation and importing step, the database contains identifiers `A.Herzan` and `A. Herzan`. These two identifiers have 12 and 1 publication(s) respectively. Although there are two identifiers in the database, it is highly improbable that the presence of a space in one indicates a second author. In the Further Analysis subsection we will discuss methods to determine if the multiple identifiers represent the same author. In this subsection, the analysis described finds identifiers that are similar to one another.

Searching for similar identifiers could happen either online (immediately after the user-submits a query) or offline (before the app is presented to users). Because our database is static and manually updated with new entries periodically, the offline approach makes sense. An additional benefit to the offline approach is that it can be easily moderated and tweaked with user submitted suggestions. The general problem is referred to as approximate string matching. If the supplied query was `A.Herzan`, then `A. Herzan` would be considered an approximate string match. This type of match could be found without much sophistication. However, we want to also consider more difficult matches like `J.Svenne` and `J.P.Svenne`. Approximate string matching libraries often use the Levenshtein distance metric to compare strings [26].

5.2.1 Levenshtein Distance

String edit distance measures such as the Levenshtein Distance [27] offer an easy first approach to analyzing the author names. The Levenshtein distance is one type of string metric to evaluate the difference between two sequences of characters. A distance of 1 is attributed to every single character edit necessary to transform one of the input strings into the other. Single character edits include an insertion of a character, a deletion, or a substitution.

The Python library Jellyfish makes it quite easy to use a few different distance metrics. Nevertheless, calculating any measure for all pairs of authors is a large task. A quick estimate of $100,000$ authors means $5,000,000,000$ unique (unordered) pairs to calculate. Thankfully this is not entirely prohibitive to calculate on modest hardware. It does, however, produce a large amount of data, making filtering absolutely necessary.

A small Python script, using Jellyfish, was prepared to calculate the Levenshtein Distance for each author name pair. Only pairs with a distance less than 4 were written to file. This resulted in over 20 million pairs. It was observed that pairs with a Levenshtein distances of 2 or greater were unlikely to be duplicate representations of the same author. Furthermore, 20 million pairs is too many for additional analysis.

5.2.2 Transformations

Three simple string transformations were constructed to locate similar identifiers. The first stage transformed all the characters in the name string to lower case. 1936 author names became non-unique when reduced to only lower case letters.

```
C.Le Brun   C.Le brun   C.le Brun
P.Fan   P.fan
A.De Waard  A.de Waard
R.Del Moral R.del Moral
J.M.Van Den Cruyce  J.M.Van den Cruyce  J.M.van den Cruyce
```

Snippet 8: Identifiers which became duplicates after transformation 1.

The second stage took the lower case identifiers and removed all spaces. There were 2619 identifiers that had duplicates when reduced to lower case letters with no spaces.

```
B.N.Subba Rao   B.N.Subbarao
R.M.Del Vecchio R.M.DelVecchio  R.M.Delvecchio  R.M.del Vecchio
J.Adam, Jr. J.Adam,Jr.
M.Le Vine   M.LeVine    M.Levine
C.Ciofi Degli Atti  C.Ciofi Degliatti   C.Ciofi degli Atti
C.Le Brun   C.Le brun   C.LeBrun    C.Lebrun    C.le Brun
```

Snippet 9: Identifiers which became duplicates after transformations 1 and 2.

Finally, we removed all punctuation as well, which resulted in 6561 identifiers that were not unique. A python script, `calc-author-name-transform-pairs.py` was prepared to perform these transformations and write the identifiers which form duplicates to a file.

```
B.V.T.Rao   B.V.Trao
A.M.Laird   A.M<.Laird
H.-R.Kissener   H.R.Kissener
W.-X.Huang  W.-x.Huang  W.X.Huang
C.Le Brun   C.Le Brun,  C.Le brun   C.LeBrun    C.Lebrun    C.le Brun
```

Snippet 10: Identifiers which became duplicates after transformations 1, 2, and 3.

As the progression of transformations shows, an identifier that becomes non-unique in transformation 1 will continue to appear in the output results of transformations 2 and 3. Some authors have been represented up to 6 different ways. Surnames composed of multiple words separated by spaces are likely to be multiply represented. The output of transformation 3 provided a list of reasonable size to apply additional analysis. There are 3063 groups of identifiers identified as duplicates in the transformation 3 analysis (and 6561 identifiers in total).

We have reduced, by two orders of magnitude, the number of identifiers that should be subject to additional analysis. With the transformed list, it is worth repeating analysis. Performing the Levenshtein distance analysis on the 'nopunc' list will locate identifiers where an initial has been omitted as an edit distance of 1. For example the edit distance of `J.P.Svenne` and `J.Svenne` is 2 before the transformations and 1 afterwards.

Performing the Levenshtein distance analysis will still fall short of identifying identifiers where the first name is fully spelled out. `Adam Sarty` and `A.Sarty` are both valid identifiers for a single author. An application to locate multiple identifiers of this type would require a significant modification to the existing string metrics. There are many open source implementations of string distance functions, so a modification is not out of the question. However, such modification is outside the scope of this work.

5.2.3 Collaboration Groups

In addition to single authors who may or may not be multiply identified, there are collaboration groups. There are 1359 identifiers that include "`the`" in their name. Identifiers representing collaborations are often long with an acronym as the informative part of their name. Table 11 shows that Levenshtein distances of 2 or greater are likely to be different collaborations.

```
For the CMS Collaboration   for the 8B Collaboration    4
For the CMS Collaboration   for the A1 Collaboration    4
For the CMS Collaboration   for the A4 Collaboration    4
For the CMS Collaboration   for the AMS Collaboration   2
For the CMS Collaboration   for the BES Collaboration   3
For the CMS Collaboration   for the CBM Collaboration   3
For the CMS Collaboration   for the CDF Collaboration   3
For the CMS Collaboration   for the CE71 Collaboration  4
For the CMS Collaboration   for the CERES Collaboration 4
```

Snippet 11: Levenshtein distances > 1 on collaborations.

5.3 The Application

The cosine similarity results are presented in the web application via the "Similar Papers" view. The user of the application can search for an author and see entries that are similar to the entries the author has coauthored.

A two-staged database query is then performed. We first get all of the similar paper `_id`s from the `simPapers` array in each of the inputted author's entries. The total list of similar paper `_id`s can be filtered by the similarity score that is also included in the `simPapers` array. At this stage we have a list of `_id`s that are similar to one or more NSR entries the inputted author published. We fetch the NSR entry for the full list of `_id`s and filter out any that were published by the inputted author. The similarity ranking considers the selectors used, and authors often publish multiple times using similar selectors, and as a result, the recommended entries often include publications from the same author(s) as the inputted paper.

The render object is then prepared to be sent to the html template to show the user. The end user then sees a web page with the search author in prominent text followed by a list of entries that have a cosine similarity to at least one of their own entries greater than 0.65. An example for author "R.A.E.Austin" is shown in Figure 7. The similar entries are sorted in descending order of their score function value. The scoring function is the average of all the `score` fields for that paper that were encountered in the aggregation.[16]

5.4 Future Work - Text Mining

Topic Modelling is a statistical model used to discover abstract topics in a collection of text [28]. A commonly used topic modelling algorithm is Latent Dirichlet Allocation (LDA) [29]. It models documents as having been created by sampling a distribution of topics [30]. The topics are distributions of words. This approach has proven effective on natural language text as well as on other data sources [31]. The selectors present in NSR entries are not natural language text but they could be used as input to LDA. This could find "topics" in the NSR data where topics are made up of NSR selectors such as nuclear isotopes and types of measurements. The Gensim package comes with topic modelling modules and algorithms, such as LDA, that could be used in future work.

The product of the analysis in the section on Transformations is a list of candidates which may represent cases of multiple identifiers for a single author. To confirm multiply-identified single authors, that candidate list must be examined. Since the list is comparatively small we propose computationally expensive analyses may be performed to achieve that end. One method could use the network graph information and compare neighbors in the network [32]. We could find all the neighbours of two given nodes and see how many are common to both. With this we should also consider what the chance of having common neighbours is for any two random nodes. A first approximation would be to consider the degree of the neighbours. Common neighbours with a low degree are less likely to be common through random chance.

[16]Because we fetch the `simPapers` array for multiple entries when searching for an author's similar entries, we can see the same `_id` multiple times and with different scores each time.

6 Data Mining

The ultimate goal of the analysis in this section and the Text Mining section is to create a flexible system that can recognize objects that are similar and not the same. Support for different types of objects within the database has been implemented. The work in the Similar Papers section enabled finding papers that had similar nuclide selectors associated with them. In this section we use association mining to produce lists of association rules that could be used in a future recommendation system. Additionally this analysis enables finding similar authors based on clustering attributes of their publication traits. An obvious use case of this feature for nuclear scientists is to find similar authors to those the user is currently inspecting or searching. Experts from other domains will be able to use these developments in other use cases.

Implementing this feature requires a significant amount of offline data mining and analysis. Once the analysis is done, the runtime of the application need only do quick lookups in tables to find the desired results. With this in mind, the high level summary of this analysis stage is to build data labels and relationships and then enable the user interface to search and display the results.

6.1 Association Mining

Frequent pattern mining is an important part of data mining and knowledge discovery [33]. It is also known as rule learning and is frequently used on market basket analysis. A history of customer transactions at a supermarket is analyzed to find groups of items that are frequently purchased together. For example, when customers purchase item A, the frequently purchase item B in the same transaction. The connection is between the items A and B and is independent of customers.

Our analysis will make use of the Apriori algorithm implementation in the arules[34] [35] package in R [36].

Association rules are similar to if-then constructs. A rule written {R (N,G),T 238U} => {N 239U} with support 0.00129 and confidence 0.9308 tells us that the selectors `R (N,G)`, `T 238U`, and `N 239U` appear together in 0.129% of the data. The confidence is a measure of reliability in the rule. In the above example 93.08% of the time that `R (N,G)` and `T 238U` appear, `N 239U` also appears. Formally the support is defined [33] in Equation 8, as is confidence in Equation 9. Lift is defined in Equation 10 as described in the `arules` package [34].

$$\text{support} = \frac{\text{count}\,(X \cup Y)}{n} \tag{8}$$

$$\text{confidence} = \frac{\text{count}\,(X \cup Y)}{\text{count}\,(X)} \tag{9}$$

$$\text{lift} = \frac{P\,(X \cup Y)}{P\,(X)\,P\,(Y)} \tag{10}$$

The `prepare-data.py` program generates three transaction lists for analysis in R. The R script `Apriori-dedup.r` takes an input file, output file, minimum support, and minimum confidence as command line arguments. The `arules` package provides facilities in helping prune duplicate rules, and rules that are subsets of other rules. However, in analyses that produce many thousands of rules, such as those using low minimum support thresholds, this pruning is computationally expensive and thus not used. As a result, the final list of rules on our extended runs contained many duplicates.

Our first analysis will use each NSR entry as a transaction, and the itemset will be the list of authors for that NSR entry. The resulting rules will be made up of authors who frequently publish together. Since Apriori finds frequent item sets, this analysis will favour authors with many publications in the NSR (and thus appear frequently). If we specify a minimum support of 0.0008, Apriori yields 344 association rules involving 104 unique author identifiers. Table 9 shows a sample of the resulting rules.

Table 9: Frequent itemset rules for authors of entries.

rules	support	confidence	lift
{F.Scarlassara,L.Corradi} => {G.Montagnoli}	0.0008287	0.9813	1022.4
{A.M.Stefanini,L.Corradi} => {G.Montagnoli}	0.0008129	0.9687	1009.2
{A.M.Stefanini,F.Scarlassara} => {G.Montagnoli}	0.0008077	0.9625	1002.7
{A.M.Stefanini,F.Scarlassara} => {L.Corradi}	0.0008025	0.9562	974.9
{G.G.Adamian} => {N.V.Antonenko}	0.0008182	0.9397	942.9
{H.Iwasaki,S.Shimoura,S.Takeuchi} => {T.Minemura}	0.0008497	0.9257	928.9
{L.Corradi} => {G.Montagnoli}	0.0008549	0.8716	908.1

All of the 11 unique authors in Table 9 have published more than 165 times. There are only 608 authors who have greater than 165 publications in the database. In order to have rules involving more authors[17] we need to lower the minimum support. The minimum support to see a given author in a rule is dependent on their publication count. If an author has published 65 times, support less than 65/212835 is required to include any rule that involves them. The author still needs to have a rule that satisfied the confidence constraint as well.

On an extended run with a low support of 0.00029 the Apriori algorithm produces 2.2 million rules. With this many rules we can no longer prune duplicates in R as the memory requirements are enormous. However we can perform some simple analyses like counting unique authors. With a support value of 0.00029 the analysis of author lists from NSR entries produces 2211797 rules involving

[17]Recall we have on the order of 100,000 authors in the database.

859 unique author identifiers.

Applying apriori with each paper as a transaction and selectors as items should produce lists of selectors that frequently occur together in NSR entries. This tends to produce association rules that look like a list of isotopes involved in nuclear reactions. The first four rules in Table 10 can be read off as nuclear reactions. ^{290}Lv undergoes alpha decay and produces the daughter nucleus ^{286}Fl, along with an alpha particle but this is not recorded in the NSR selectors. We see that ^{290}Lv and alpha decay are never mentioned without ^{286}Fl. With a support value of 0.00029 the analysis of selectors from their NSR entry lists produces 3479553 rules involving 1202 unique selectors.

Table 10: Frequent itemset rules for selectors in NSR entries.

rules	support	confidence	lift
{P 290LV,S A-DECAY} => {G 286FL}	0.000298	1.0000	3346.7
{P 289FL,S A-DECAY} => {G 285CN}	0.000310	1.0000	3225.0
{P 29418,S A-DECAY} => {G 290LV}	0.000310	1.0000	3225.0
{P 286FL,S A-DECAY} => {G 282CN}	0.000310	1.0000	3225.0
{G 285CN} => {P 289FL}	0.000310	1.0000	3167.4
{P 289FL} => {G 285CN}	0.000310	0.9821	3167.4
{G 286FL} => {P 290LV}	0.000298	1.0000	3167.4
{G 285CN,S A-DECAY} => {P 289FL}	0.000310	1.0000	3167.4
{G 286FL,S A-DECAY} => {P 290LV}	0.000298	1.0000	3167.4
{P 290LV} => {G 286FL}	0.000298	0.9464	3167.4

Our final analysis with Apriori uses each selector as a transaction and the list of authors who have published with that selector as the itemset. Some of these rules may involve authors who have not published together. This information would be useful, however the analysis to find such a rule has not been completed.

With a support value of 0.0042 the analysis of author lists for each selector produces 3832412 rules involving 774 unique author identifiers.

If we want to find authors who have not published together but do publish on similar keywords, this analysis is not optimal. A more efficient approach would leverage the graph data in Nuclide Graphs. The selector rules found above could be used to enlarge the search query for a graph. So instead of searching for just `290LV` we could enlarge the search by also including entries with `286FL`. Alternatively we could reduce the search results by including only entries with both `290LV` and `286FL`.

Table 11: Frequent itemset rules for selectors in entries.

rules	support	confidence	lift
{B.Kindler} => {B.Lommel}	0.008280	0.9733	110.0
{W.G.Lynch} => {M.B.Tsang}	0.008723	0.9236	100.7
{V.I.Chepigin} => {A.P.Kabachenko}	0.008846	0.9148	96.7
{E.Fioretto,L.Corradi} => {S.Szilner}	0.008043	0.9536	93.6
{A.Gadea,A.M.Stefanini,G.Montagnoli} => {S.Szilner}	0.008012	0.9487	93.1
{A.Gadea,F.Scarlassara,G.Montagnoli} => {S.Szilner}	0.008125	0.9437	92.6
{A.Gadea,G.Montagnoli,L.Corradi} => {S.Szilner}	0.008115	0.9425	92.5
{F.Scarlassara,L.Corradi,S.Szilner} => {G.Montagnoli}	0.008517	1.0000	92.4
{A.M.Stefanini,L.Corradi,S.Szilner} => {G.Montagnoli}	0.008403	1.0000	92.4

6.2 Cluster Analysis

Classification and clustering are related approaches to organizing data elements into groups for further analysis. Classification is the process of deciding to which group a particular datum should most optimally belong. Clustering is the grouping of multiple data points such that those belonging to a group are more similar in some manner than those outside of that group.

6.2.1 K-means Clustering

K-means clustering is a cluster analysis technique that can group data objects in k clusters based on minimizing their distances with respect to cluster centroids [37] [38] [39]. K-means is a partitional clustering algorithm.

Take a finite set of objects, $X = x_1, x_2, ..., x_n$ where each is a data object in d dimensions. We can create k clusters $C = c_1, c_2, ..., c_k$ where $k <= n$. The process starts by randomly choosing k points, $x_1, ..., x_k$ to be the centroids of a cluster. The process continues by iterating over each object x and assigning it to a cluster c based on the minimization of some parameter; for now, Euclidean distance. The new centroids are then computed and the process is repeated until cluster stability is achieved. The goal is to minimize the total sum of squared errors between the centroids and all objects (see Equation 11).

$$J(C) = \sum_{k=1}^{K} \sum_{x_i \in c_k} |x_i - \mu_k|^2 \tag{11}$$

Three parameters for K-means must be specified initially. The number of clusters, initial centroid guesses, and the distance metric. The metric is the function on a space that describes how two points differ from one another, i.e. distance. Euclidean distance is typically used, leading to ball-shaped or sphere-shaped clusters. [38]

The chosen number of clusters has a huge impact on the data partitions. Some heuristics exist to aid in determining an optimal k. [40] In practice, K-means is normally run multiple times with varying k values and the optimum is selected by a domain expert.

However, there exist methods to measure the effectiveness of a clustering con-

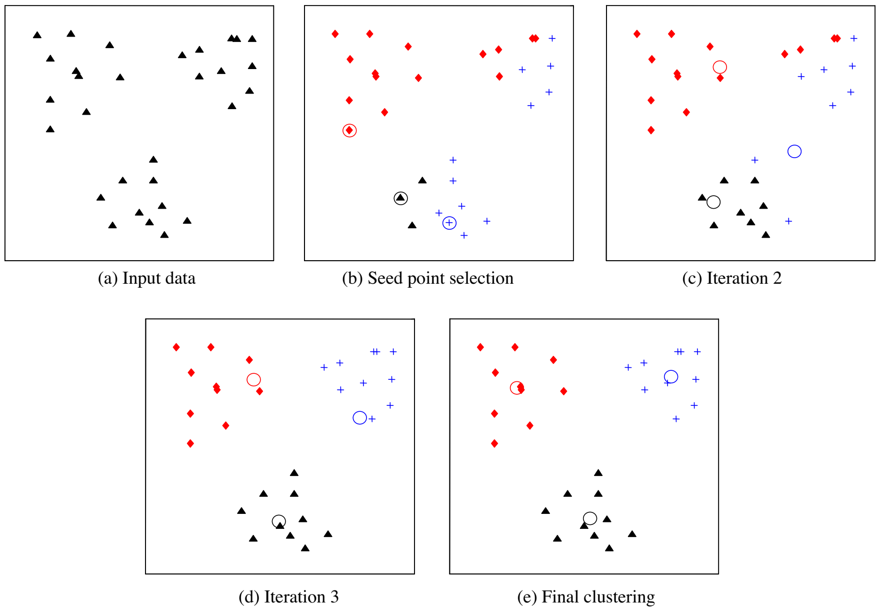


Figure 17: Step by step illustration of K-means algorithm. (a) The initial input data. (b) Three seed points are chosen as the starting 'centroids' and the data points are assigned to the cluster with the closest seed point. (c) (d) The centroids of the new clusters are calculated, and data labels updated; (e) the iteration stops when the clusters converge.

figuration. The Davies-Bouldin index considers the ratio of external separation between clusters to the scatter within a cluster [41] [42]. Given two clustering schemes of the same input data, the one with the lowest Davies-Bouldin index is preferred. The G1, or Calinski-Harabasz criterion is a hueristic device to help evaluate different clustering schemes on the same input data [43]. It works best when used on standardized data in a Euclidean space [44]. In contrast to the Davies-Bouldin index, a higher G1 index value suggests a better clustering scheme. Both of these evaluation methods are provided in the R package `clusterSim` [45].

6.2.2 Initial Author Clustering

There is a considerable amount of multivariate data in the NSR database. In order to gain insight from this data, only a section is initially considered. In this section, the authors will be analyzed to identify groups or clusters based on publication traits. Specifically, we considered three parameters: the total length in years an author has published to date, their average number of coauthors across all their NSR entries, and their total publication count.

To help evaluate this summarization of authors, three heat maps of this data are presented in figures 18, 19, and 20. Figure 18 shows an expected trend: Authors who publish over more years tend to have more publications overall. Each of these figures shows that there are a large number of authors who have published only a few times. Additionally, there are comparatively few authors who have published many times. The heat map colouring is on a logarithmic scale, while the axes are linear. Without the log colour scale, the plots would be washed out by the incredibly many authors who have published only once. (Linearly coloured heat maps are shown in the Appendix, figures 27, 28, and 29.)

The three heat maps show that the 3 dimensional data are not well segmented and are instead continuous. This is a result of the input data being continuous

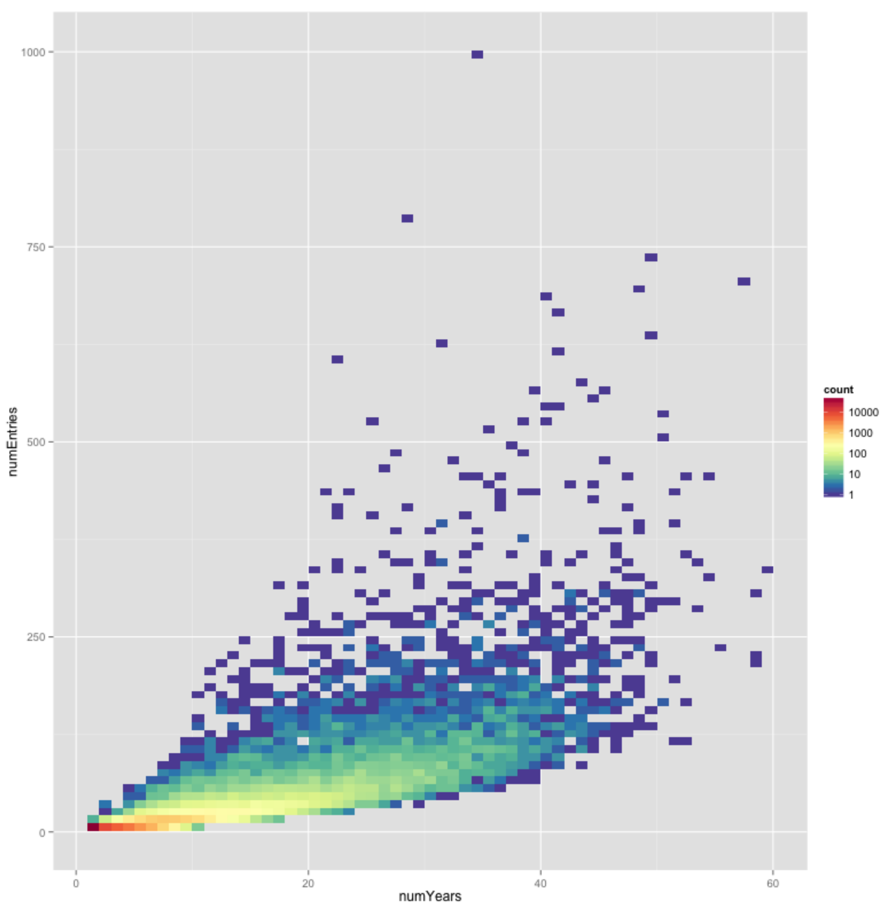


Figure 18: Number of entries from an author related to the number of years the author has published.

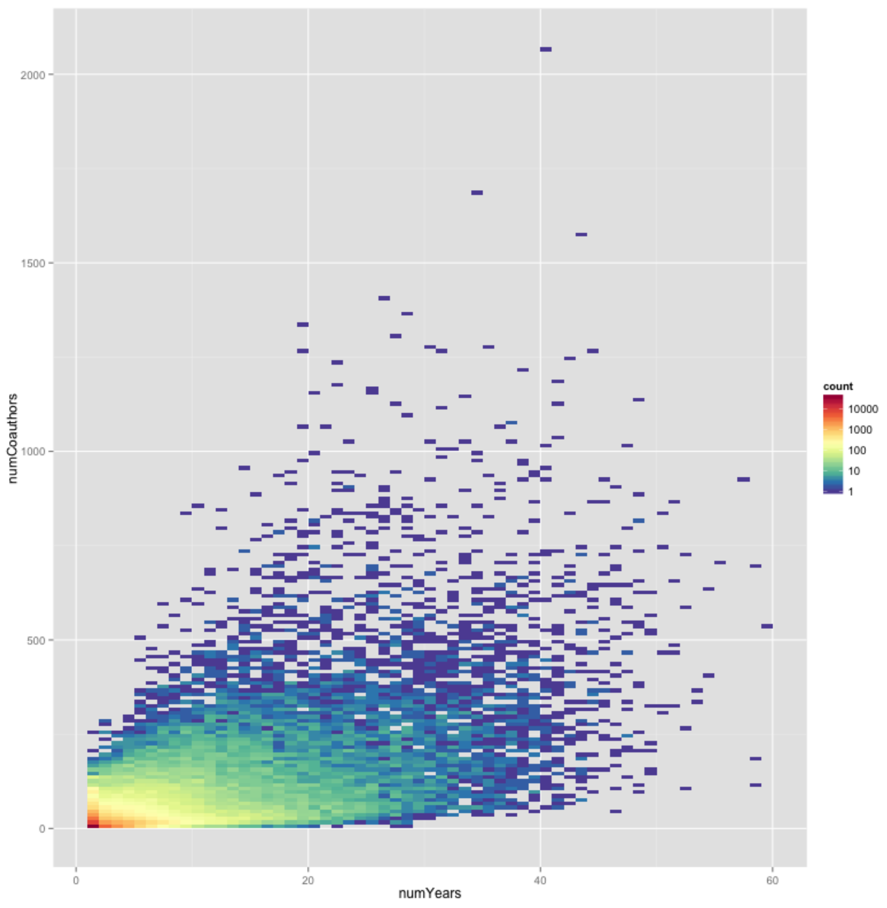


Figure 19: Number of coauthors associated with an author given the number of years the author has published.

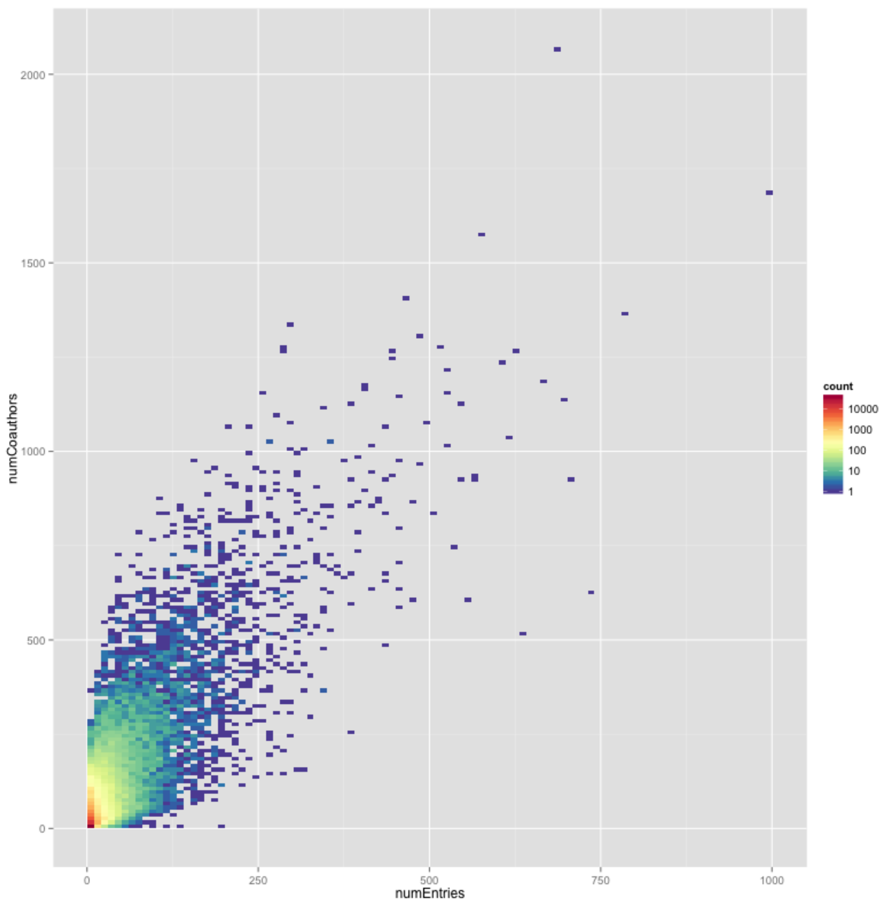


Figure 20: Number of coauthors associated with an author given the number of entries the author has published.

in nature. Clustering categorical data could lead to more discrete or separated clusters. Nevertheless, the cluster results of this data could be of value to researchers using these tools.

The Davies-Bouldin index and G1 index have been calculated for all K-means clustering schemes from 2 centers to 16. The results are plotted in Figure 21 for Davies-Bouldin index and Figure 22 for the G1 index. The Davies-Bouldin index suggests either 5 or 6 cluster centers is best for this data. The G1 index results suggest that 5 cluster centers is best, with 6 being the next best.

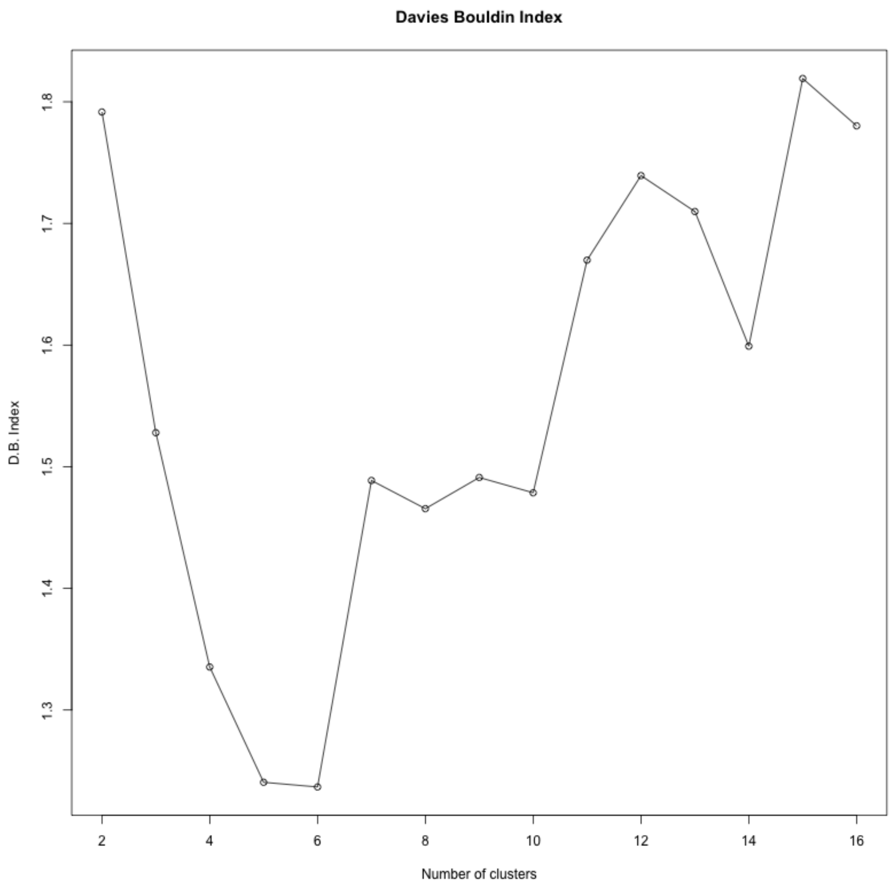


Figure 21: Davies-Bouldin index for number of clusters

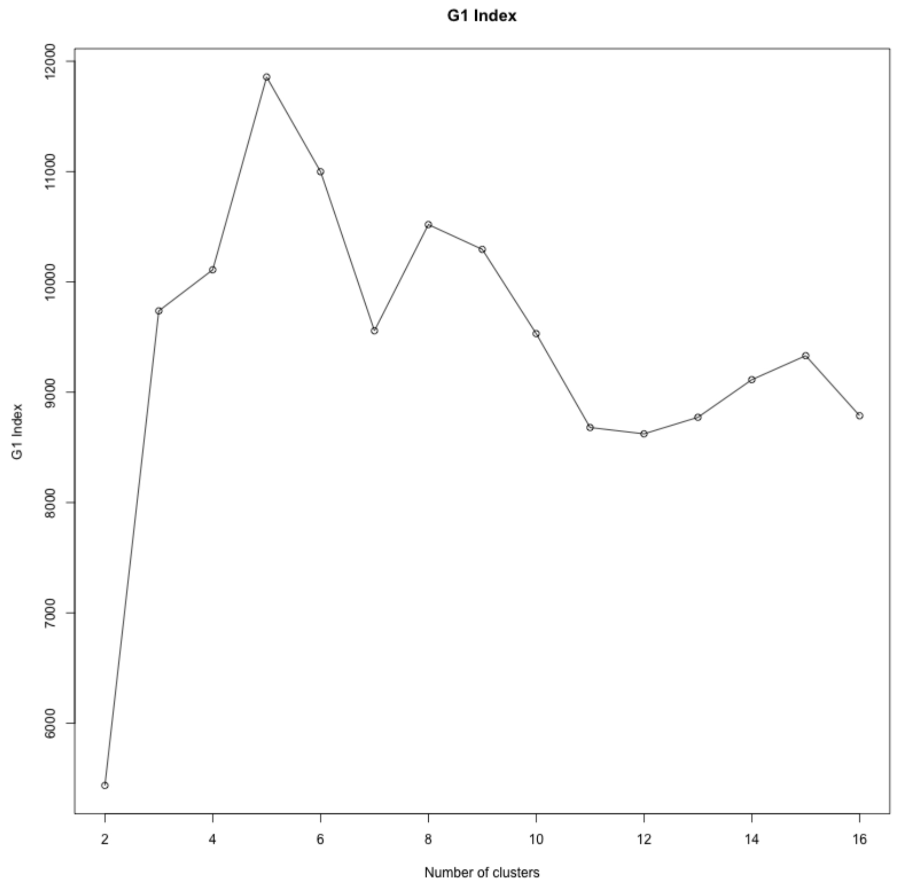


Figure 22: G1 index for number of clusters

The two cluster evaluation metrics suggest that 5 or 6 cluster centers is optimal. Table 12 shows the data point values for the 5 different cluster centroids. Table 13 shows the centroid information for the 6 cluster scheme. Note that the data has been standardized, so the values in the tables are in standard deviations.

Figure 23 shows the `numCoauthors`, `numYears`, and `numEntries` data coloured according to their cluster membership in the 5 cluster scheme. This figure again demonstrates that the data are continuous and that well separated clusters do not exist in this domain. As a result, the clusters function as segmentations along a continuous spectrum. As the number of clusters increases, the size of the segmentations decreases.

Table 12: Centroid data points for 5 cluster K-Means on initial data

careerLength	meanCoauthors	numEntries	size
-0.67901	1.68445	-0.03426	2342
0.91890	-0.44386	-0.12959	5510
0.87978	2.90728	4.92022	316
-0.70500	-0.34365	-0.40397	8262
1.31926	0.26803	1.63938	1572

Table 13: Centroid data points for 6 cluster K-Means on initial data

careerLength	meanCoauthors	numEntries	size
-0.69065	0.91707	-0.21925	3131
1.20883	2.50212	5.70388	231
1.29734	0.31428	1.68485	1537
-0.67117	-0.48459	-0.41750	7213
0.96457	-0.44891	-0.11525	5301

careerLength	meanCoauthors	numEntries	size
-0.64999	3.29831	0.68203	589

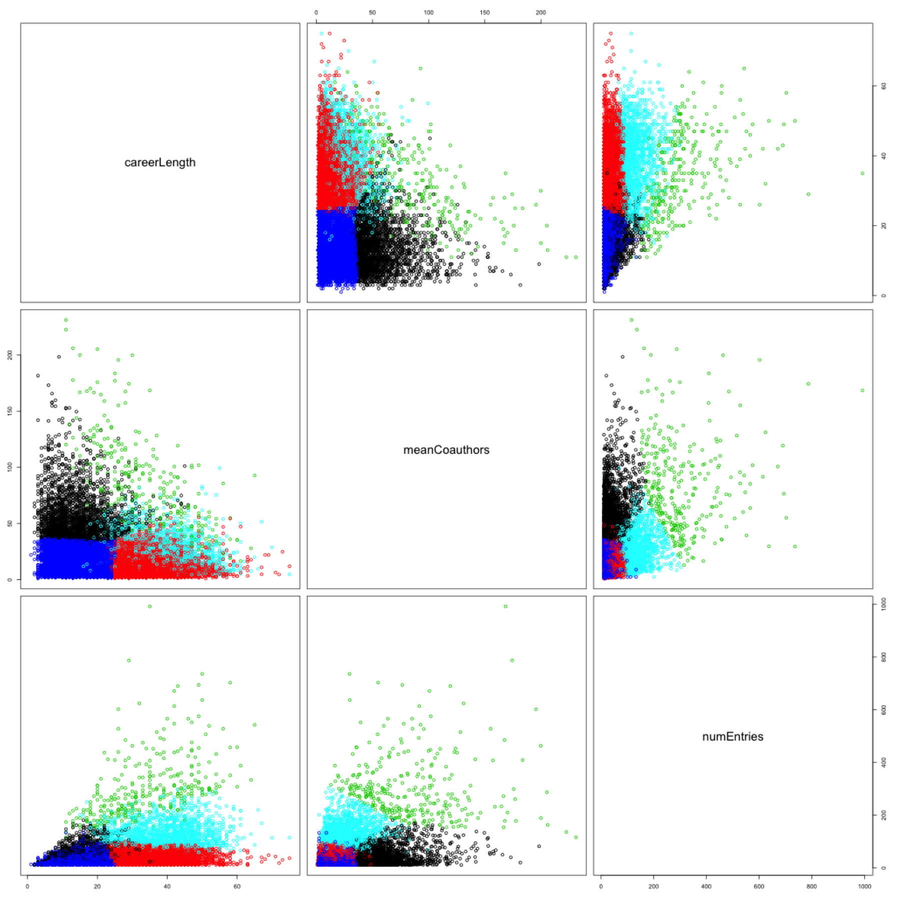


Figure 23: Initial clustering on authors with K-means

The first cluster in Table 12 represents authors who have had short careers but published an average number of entries with above-average numbers of coauthors. The second cluster of authors have had long careers and published an average amount with an average number of coauthors. The authors in the third cluster have published the most and frequently publish with many coauthors. The fourth and largest cluster of authors are those with shorter careers and fewer publications with fewer coauthors. Finally, the fifth cluster of authors have had the longest careers, published many times, and overall, have published with an average amount of coauthors.

6.2.3 Secondary Clustering

In this analysis, an author publishing 20 entries in 1995 and 5 entries in 1997 has a numYears value of 3. The previous section's analysis was blind to how prolific a given author may have been in shorter periods. Our goal is for the the author who published 20 entries in 1995 to be measured differently than an author who published once in each of 1995 and 1997.

Instead of a single number representing an author's number of publications, a parameter representing the distribution of publications over time is used. How many entries did a given author publish in the beginning of their career? How many entries did the author publish at the end of their career, or most recently? We take three measurements for each author regarding their publication history: the percentage of their total publications in the first, second, and final third of their career to date.

In order to break up the number of entries over time like this, each author needs to have multiple entries over multiple years. Recall there are 41254 authors with only a single publication in the database. That is 41254 out of 100147, roughly 40% of the total unique authors. Therefore we lose 40% of the authors in the database if we require an author to have published over multiple years. In order to have a non-zero number of publications in each third an author must have a minimum of 3 publications in three different years. The requirement for publishing 3 or more times in any years cuts out 55% of authors in the database.

Recall the analysis in Author Contributions suggested that a small portion of the authors contribute a large portion of the NSR entries. This result means that filtering out low-publication authors in additional analyses does not affect the majority of the NSR entries. As a result the secondary clustering only considered the authors who contribute 90% of the NSR entries with author fields. Explicitly, we considered authors who published 11 or more times. This amounted to 18006 authors. The code to produce the input data for the secondary clustering is available in the `prepare-data.py` script.

The G1 index for the secondary clustering is shown in Figure 25. It suggests a clustering scheme of either 4 or 6 centers. The Davies-Bouldin index, shown in Figure 24 suggests either 2, 5, 6. The clustering results for 6 centers are shown in Figure 26.

Table 14: Centroid data points for 6 cluster K-Means on initial data

cLength	mCoauthors	nEntries	nEntries1	nEntries2	nEntries3	size
1.16972	-0.27103	0.46882	-0.45377	0.71277	-0.11090	3299
1.08866	1.88560	4.07120	-0.78269	0.51341	0.41716	564
0.41893	-0.21279	-0.17738	-0.83277	-0.83916	1.62115	2993
-0.64822	-0.31975	-0.36245	-0.25189	0.63417	-0.26423	5227
-0.71602	1.98927	0.10059	-0.02918	0.25118	-0.18162	1651
-0.27094	-0.26837	-0.37100	1.35796	-0.90415	-0.71240	4268

The clustering results are tabulated in Table 14. In this table, `cLength` represents the career length and `mCoauthors` the mean of the number of coauthors. `nEntries` `nEntries1` `nEntries2` `nEntries3` represent the total number of NSR entries, the amount in the first, second, and final third of an authors career respectively. Finally `size` is the number of authors in a given cluster.

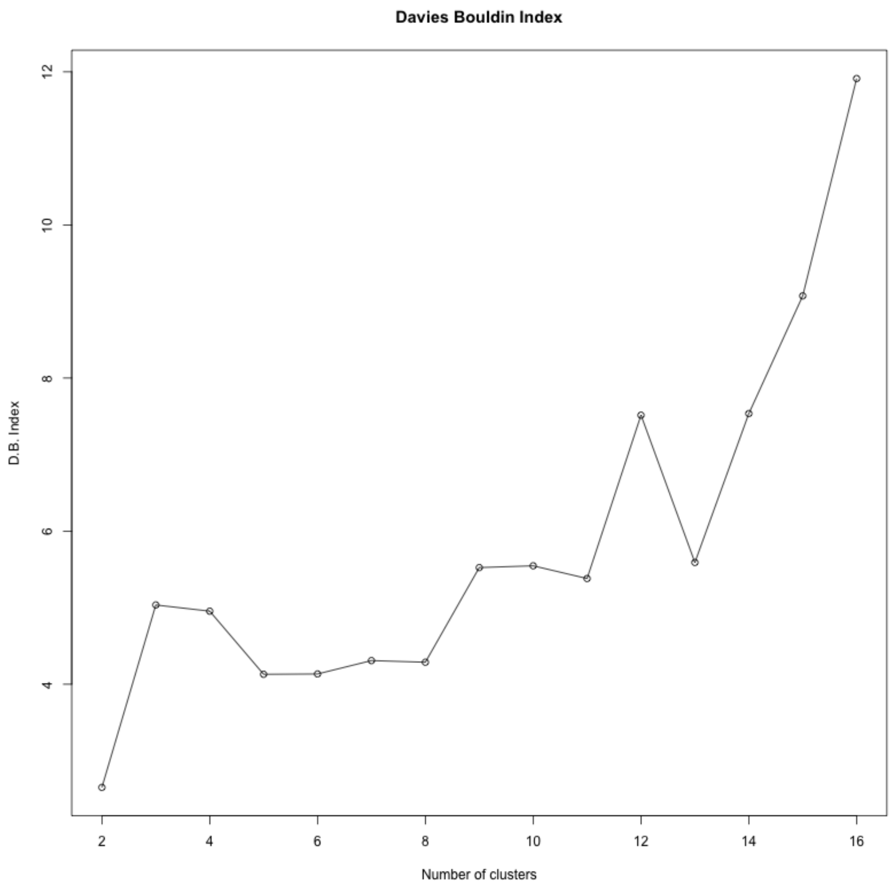


Figure 24: Davies-Bouldin index for secondary clustering

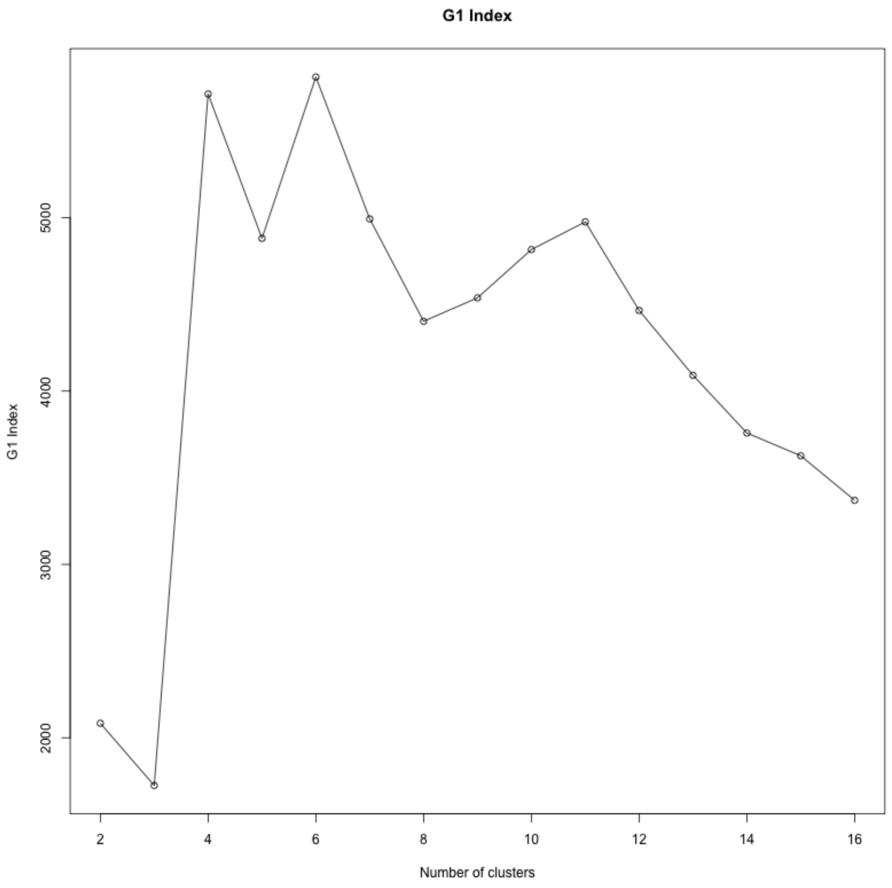


Figure 25: G1 index for secondary clustering

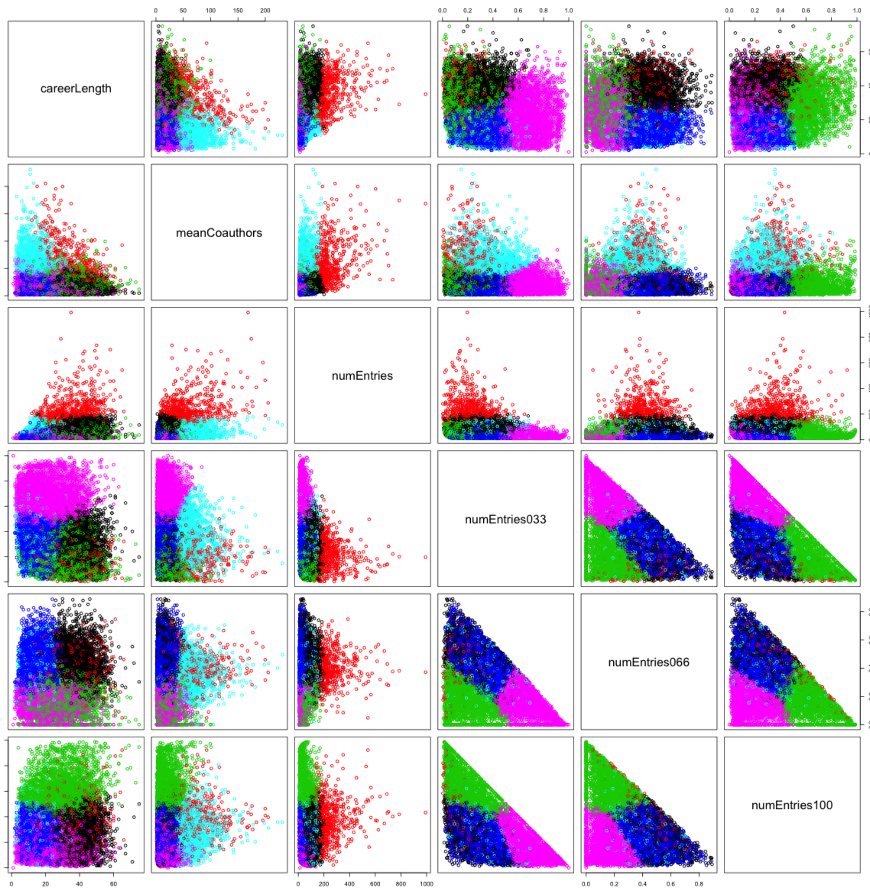


Figure 26: Secondary K-means clustering with 6 centers

The first cluster presented in Table 14 represents authors who have had longer careers, published with fewer coauthors, and published a relatively normal amount of NSR entries, mostly in the middle of their careers. The second cluster of authors had similar length careers and published with more coauthors and contributed many more entries in total. These are the most prolific of authors, and represent a small group of people, with only 564 members. The authors in the third cluster have high publication numbers in the last third of their careers. Most other features of these authors are typical. The fourth and largest cluster are those with few publications over few years mostly in the middle of their careers. The fifth cluster of authors have the shortest careers, but publish with the most coauthors on average. The sixth and second largest cluster is similar to the fourth except these authors had strong early years in their career, as well.

Note that there is currently no accounting for whether a career has ended years ago, or if the author is still actively publishing. Therefore, in this analysis, every author's career either ends naturally on some year, or artificially in 2014 when the data stops.

6.2.4 The Application

The results of cluster analysis can be written to the database with the `update-database.py` script. The clusters an author belongs to are used to colour the nodes on network graphs. This is demonstrated in Figure 5.

6.3 Future Work - Data Mining

As the NSR database spans several decades, each data object presents time series information. Finding similar authors separated in time could be interesting. Additionally there is a dramatic change in the rate of publications over time. The average number of coauthors in 1940 may be different from the average number in later years. Weighting author publication traits in time could produce better results.

The results from the association rule learning could be used to develop a classification system. As authors input the keywords for their paper, the system could try to match the user's input with association rules. This would require relating the association rules to the desired classification rules. Alternatively, the NSR entries could be classified manually and then we could rerun Apriori to learn rules that directly link to the classification label.

7 Conclusions

This work has enabled exploration and manipulation of the NSR dataset that was inaccessible or impossible using the existing applications. The new tools and web application provide interactive search and visualizations to aid data exploration. Further research into the NSR is now more accessible by nonexperts, or experts of other domains such as network analysis or social science.

The treatment of the data as a network is a new and flexible contribution. It can be used to visualize an author's collaborators, or to see the collaborations that exist around a given topic. Accessing the data as a network can facilitate further work in studying the NSR as a social network.

The data mining analysis revealed naming issues in the database, and methods have been proposed for mitigating them. The use of text mining tools has enabled the creation of a paper recommender system that recommends based on semantic information of the given journal titles, theses, conference proceedings, and other sources. Due to the high quality of the Nuclear Science References database and in particular the selectors, this recommender system can function without access to the whole text for each recommended source.

The metadata provided by the Nuclear Science References has enabled a custom exploration and information retrieval system that extends the capabilities of the existing system and other more general search engines.

8 Appendix

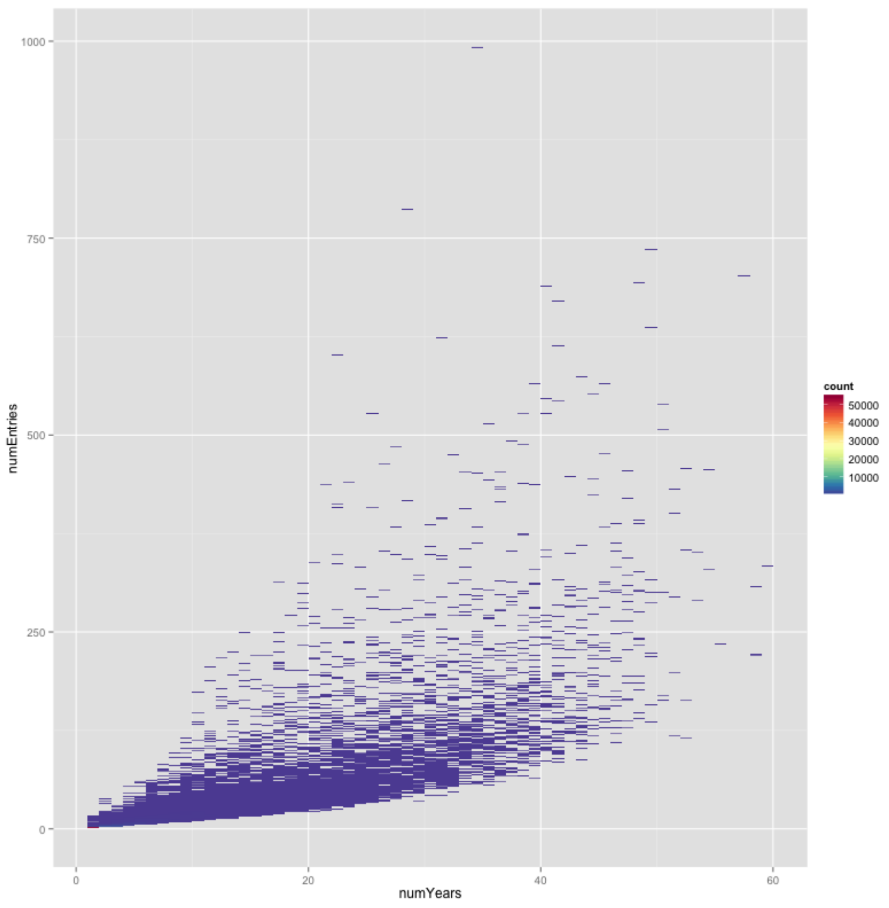


Figure 27: Number of entries contributed by an author related to the number of years the author has published.

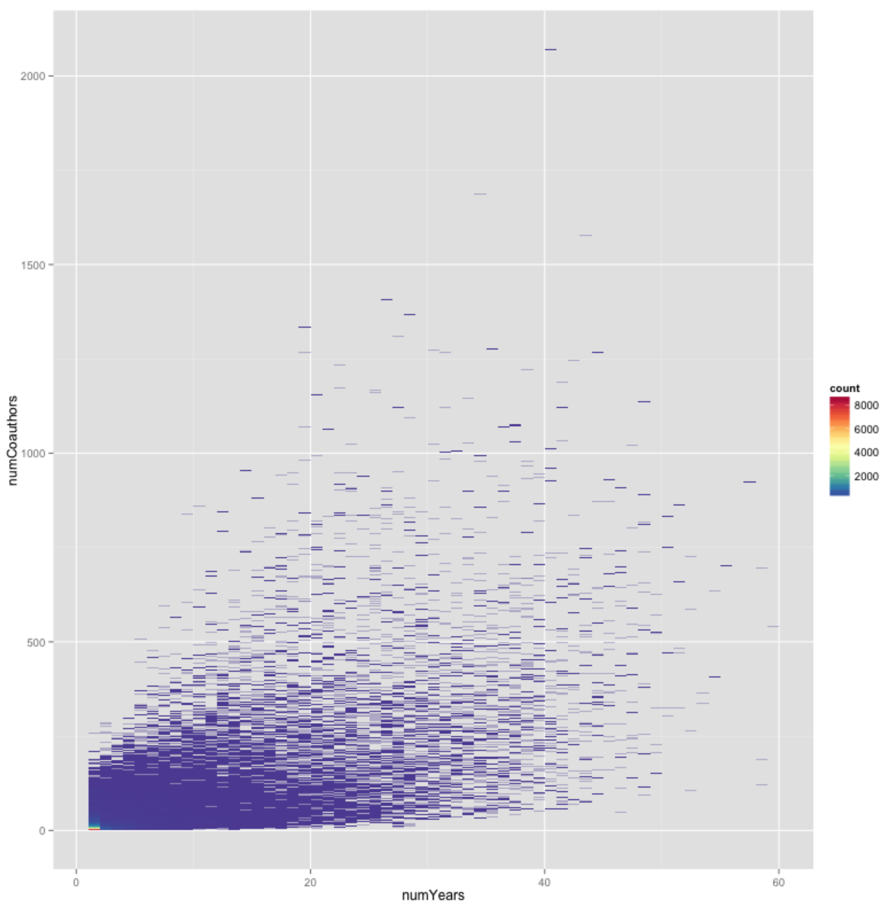


Figure 28: Number of coauthors associated with an author related to the number of years the author has published.

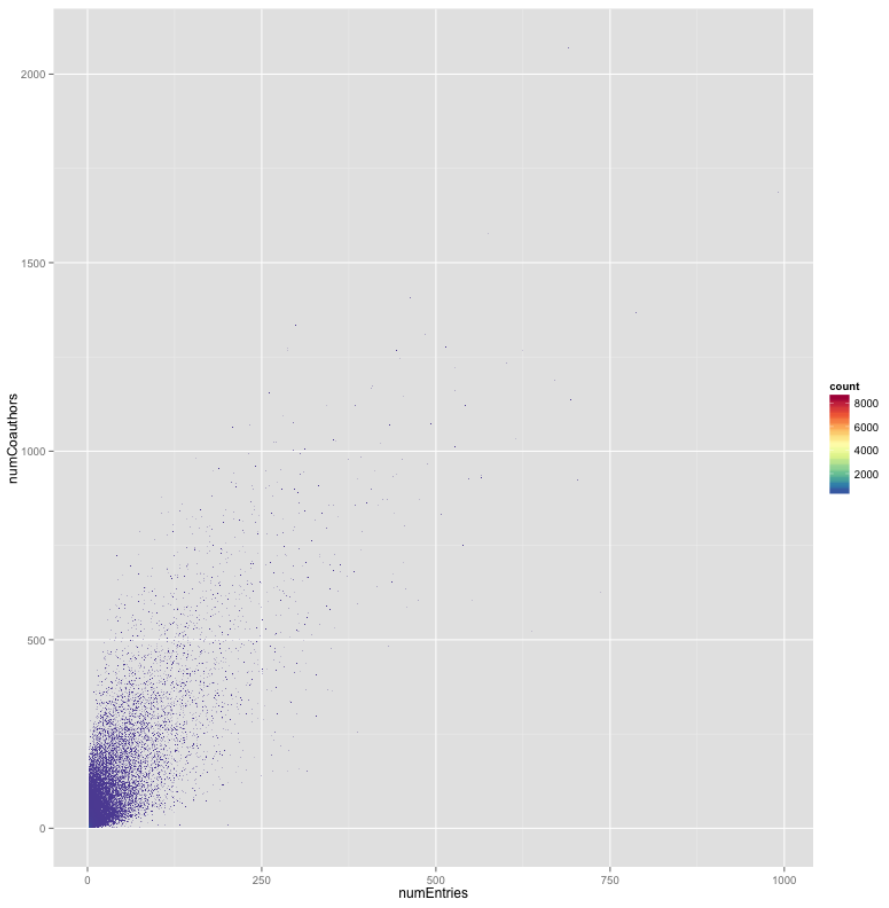


Figure 29: Number of coauthors associated with an author related to the number of entries the author has published.

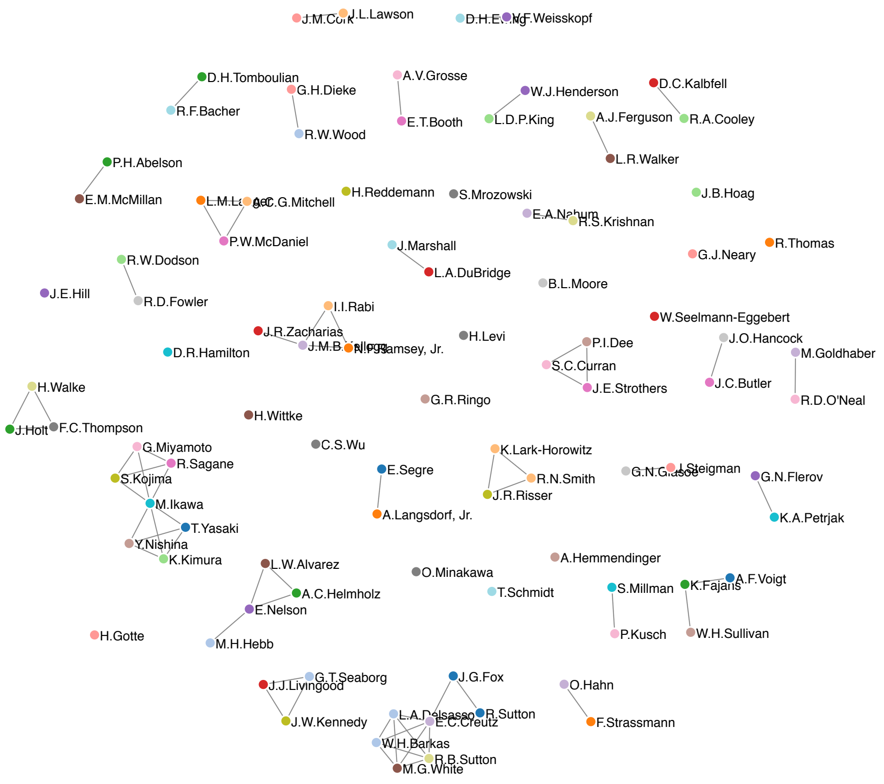


Figure 30: Complete 1940 author graph.

```
for (i=0; i<=10; i++){
    db.NSR.aggregate([
        {$project: {_id: 1, authors: 1, year: 1}},
        {$unwind: "$authors"},
        {$group: {_id: "$authors", numEntries: {$sum: 1}, papers:
 ↪  {$addToSet: "$_id"}}},
        {$match: {"numEntries": {$gte: i}}},
        {$unwind: "$papers"},
        {$group: {_id: "$papers", uniqueKey: {$sum: {$multiply: [1,
 ↪  0]}}}},
        {$group: {_id: "$uniqueKey", papersRemaining: {$sum:1}}}
    ], {allowDiskUse: true})
    .forEach(function(myDoc) {
        print( "user: " + myDoc.papersRemaining ); }
    ) }
```

Snippet 12: The mongoshell code to determine the results shown in Table 6

The code used to prepare, manipulate, and transform the NSR dataset is included in this appendix. The organization loosely follows the order one might follow to recreate this work. Note that the most updated version of these programs should be available at https://github.com/valencik/mastersAPSC.

8.1 parseNSRtoJSON.pl

```perl
#!/usr/bin/perl -pi

use strict;
use warnings;

# This helps with multiline regex
BEGIN {undef $/;}

#Fix erroneous verticle bars
s/\(\|\|\)/\\leftrightarrow/mg; # 2006VO15 and 2007EL04
s/\|\|V{-tb}\|\|/\\vert |V{-tb} \\vert/; # 2007AB22
s/\|\|g/\|g/mg; #2009RE20 2011KU10 2012KR07
#s/measured E\|g,I\|\|g\. Data/measured E|g,I|g. Data/; #2007MIZO
s/the \(\\{\+3\\}He,t\) Reaction/the ({+3}He,t) Reaction/;
↪  #1984VAZR
s/A\\\|'/\\AA/; #1996RA31

# Cleaning up the CODEN's
s/^<CODEN   >(JOYR|JUOUR|JOur|PRVCA|Nature|J\{OUR) /<CODEN
↪  >JOUR /mg;
s/^<CODEN   >Conf /<CODEN   >CONF /mg;
s/^<CODEN   >(REPR|REP>T|REPTT|Rept) /<CODEN   >REPT /mg;
s/^<CODEN   >Book /<CODEN   >BOOK /mg;
s/^<CODEN   >(THE{SIS|Thesis,) /<CODEN   >THESIS /mg;

# Force any remaining problems to simply be type UNKNOWN
s/^<CODEN
↪  >(?!JOUR)(?!REPT)(?!CONF)(?!THESIS)(?!PC)(?!PREPRINT)(?!BOOK)/<CODEN
↪  >UNKNOWN /mg;

# Escape double quotes
s/"/\\"/g;

# Collapse all multiline entries to single line
```

```
s/&\n(.)(?!KEYNO)(?!HISTORY)(?!CODEN)(?!REFRENCE)(?!AUTHORS)(?!TITLE)
↪  (?!KEYWORDS)(?!SELECTRS)(?!DOI)/$1/mg;

# Remove tailing whitespace and tabs
s/ +&$/&/mg;
s/\t/ /mg;
s/, ,/, /g; #Fix empty separated values

# Basic parsing
s/^<KEYNO   >(.*)&$/{\n"_id":"$1",/mg;
s/^<HISTORY >(.*)&$/"history":["$1"],/mg;
s/^<CODEN   >(.*)&$/"code":"$1",/mg;
s/^<REFRENCE>(.*)&$/"reference":"$1",/mg;
s/^<AUTHORS >(.*)&$/"authors":["$1"],/mg;
s/^<TITLE   >(.*)&$/"title":"$1",/mg;
s/^<DOI     >(.*)&$/"DOI":"$1",/mg;
s/^<KEYWORDS>(.*)&$/"keywords":["$1"],/mg;
s/^<SELECTRS>(.*)&$/"selectors":["$1"],/mg;

# Remove double whitespace
s/ {2,}/ /g;

# End JSON structures
s/,\n{/\n}\n{/mg;
s/,$/\n}/;
```

8.2 massageJSONtoSchema.pl

```
#!/usr/bin/perl -pi

use strict;
use warnings;

# Extract year to its own field
s/^"_id":"((\d\d\d\d).*)",$/"_id":"$1",\n"year":$2,/g;

# Extract type into its own field
s/^"code":"((\w*).*)",$/"code":"$1",\n"type":"$2",/g;

# Turn author names into array elements
s/([^]]), /$1", "/g if /^"authors":\[/;
s/", "Jr."/, Jr."/g if /^"authors":\[/; #Fix the Jr's
s/"authors":\[""\],/"authors":["UNKNOWN"],/g if /^"authors":\[/;
↪  #Fix empty fields

# Turn history stamps into array elements
s/(\b) (\b)/$1", "$2/g if /^"history":\[/;

# Split selectors
s/([A-Z]):(.*?);(.*?)\./{"type":"$1", "value":"$2",
↪  "subkey":"$3"},/g if /^"selectors":\[/;
s/^"selectors":\["/"selectors":\[/g;
s/}, *"]/}]/ if /^"selectors":\[/;

# Turn NSR free text fields into valid LaTeX
# Special characters in NSR according to NSR Manual
s/\|A/A /g;
s/\|B/B /g;
s/\|C/H /g;
s/\|D/\\Delta /g;
s/\|E/E /g;
s/\|F/\\Phi /g;
s/\|G/\\Gamma /g;
s/\|H/X /g;
s/\|I/I /g;
s/\|J/\\sim /g;
s/\|K/K /g;
s/\|L/\\Lambda /g;
s/\|M/M /g;
s/\|N/N /g;
s/\|O/O /g;
s/\|P/\\Pi /g;
```

```
s/\|Q/\\Theta /g;
s/\|R/R /g;
s/\|S/\\Sigma /g;
s/\|T/T /g;
s/\|U/\\Upsilon /g;
s/\|V/\\nabla /g;
s/\|W/\\Omega /g;
s/\|X/\\Xi /g;
s/\|Y/\\Psi /g;
s/\|Z/Z /g;
s/\|a/\\alpha /g;
s/\|b/\\beta /g;
s/\|c/\\eta /g;
s/\|d/\\delta /g;
s/\|e/\\varepsilon /g;
s/\|f/\\phi /g;
s/\|g/\\gamma /g;
s/\|h/\\chi /g;
s/\|i/\\iota /g;
s/\|j/\\epsilon /g;
s/\|k/\\kappa /g;
s/\|l/\\lambda /g;
s/\|m/\\mu /g;
s/\|n/\\nu /g;
s/\|o/o /g;
s/\|p/\\pi /g;
s/\|q/\\theta /g;
s/\|r/\\rho /g;
s/\|s/\\sigma /g;
s/\|t/\\tau /g;
s/\|u/\\upsilon /g;
s/\|w/\\omega /g;
s/\|x/\\xi /g;
s/\|y/\\psi /g;
s/\|z/\\zeta /g;
s/\|'/^\\circ /g;
s/\|`/\\ell /g;
s/\|\(/\\gets /g;
s/\|\)/\\to /g;
s/\|\*/\\times /g;
s/\|\+/\\pm /g;
s/\|\-/\\mp /g;
s/\|\./\\propto /g;
s/\|4/< /g;
s/\|5/> /g;
s/\|6/\\surd /g;
```

```
s/\|7/\\int /g;
s/\|8/\\prod /g;
s/\|9/\\sum /g;
s/\|</\\le /g;
s/\|=/\\ne /g;
s/\|>/\\ge /g;
s/\|\?/\\approx /g;
s/\|@/\\infty /g;
s/\|\[/\\{ /g;
s/\|\\/\\vert /g;
s/\|\]/\\} /g;
s/\|^/\\uparrow /g;
s/\|_/\\downarrow /g;

# Remove double whitespace
s/ {2,}/ /g;
```

8.3 flattenJSONforMongo.pl

```
#!/usr/bin/perl -pi

use strict;
use warnings;

# This helps with multiline regex
BEGIN {$/ = "}\n";}


# Flatten JSON for mongoimport
s/{\n/{/mg;
s/,\n"/, "/mg;
s/\n}/ }/mg;

# Escape all the backslashes
s/\\(?!")/\\\\/mg;
```

8.4 prepare-data.py

```
import csv
import os.path
import pymongo
import subprocess
import sys
import statistics
import math
import functools


def percentile(N, percent, key=lambda x: x):
    """
    Find the percentile of a list of values.
    From: http://code.activestate.com/recipes/511478/

    @parameter N - is a list of values. Note N MUST BE already
↪   sorted.
    @parameter percent - a float value from 0.0 to 1.0.
    @parameter key - optional key function to compute value from
↪   each element of N.

    @return - the percentile of the values
    """
    if not N:
        return None
    k = (len(N) - 1) * percent
    f = math.floor(k)
    c = math.ceil(k)
    if f == c:
        return key(N[int(k)])
    d0 = key(N[int(f)]) * (c - k)
    d1 = key(N[int(c)]) * (k - f)
    return d0 + d1


# Connect to the local Mongo server
try:
    client = pymongo.MongoClient('localhost', 27017,
↪   serverSelectionTimeoutMS=100)
    client.admin.command('ismaster')  # Test command to see if we
↪   can connect

    # If we can conenct, reset client to defaults
    print("Successfully connected to MongoDB...")
    client = pymongo.MongoClient('localhost', 27017)
except pymongo.errors.ConnectionFailure:
    sys.exit("ERROR: Could not connect to database, are you sure 'mongod' is running?")

# Create 'masters' database
if 'masters' not in client.database_names():
    print("Database 'masters' was not found, creating...")
else:
    print("Found database 'masters'...")
db = client['masters']

# Import NSR data
if 'NSR' not in db.collection_names():
    if not os.path.isfile("NSR.json"):
        if os.path.isfile("mastersNSRDataDump.tbz2"):
            print("Could not find database file NSR.json...")
            print("Found mastersNSRDataDump.tbz2, extracting...")
            subprocess.call(["tar",  "-xf", "mastersNSRDataDump.tbz2"])
            subprocess.call(["mv",  "NSRDump.out", "NSR.json"])
            print("Running perl magic...")
            subprocess.call(["perl", "parseNSRtoJSON.pl", "NSR.json"])
            subprocess.call(["perl", "massageJSONtoSchema.pl", "NSR.json"])
            subprocess.call(["perl", "flattenJSONforMongo.pl", "NSR.json"])
    if os.path.isfile("NSR.json"):
        print("NSR collection not found, importing NSR.json with mongoimport...")
        subprocess.call(["mongoimport", "--db", "masters", "--collection", "NSR",
                         "--type", "json", "--file", "NSR.json"])
    else:
        sys.exit("ERROR: Could not find data files NSR.json or mastersNSRDataDump.tbz2")
print("Found NSR collection...")

# Ensure indexing for NSR collection
if 'NSR' in db.collection_names():
    db.NSR.create_index("year")
    db.NSR.create_index("authors")
    db.NSR.create_index("type")
    db.NSR.create_index("selectors.type")
    db.NSR.create_index("selectors.value")
```

```

# Create collection authorSummary
if 'authorSummary' not in db.collection_names():
    print("Collection authorSummary not found, creating...")
    db.NSR.aggregate([
        {"$project": {"_id": 1, "copyauthors": "$authors",
↪   "authors": 1, "year": 1}},
        {"$unwind": "$authors"},
        {"$unwind": "$copyauthors"},
        {"$group": {"_id": "$authors", "coauthors": {"$addToSet":
↪   "$copyauthors"},
                    "years": {"$addToSet": "$year"}, "papers":
↪   {"$addToSet": "$_id"}}},
        {"$out": "authorSummary"}
    ], allowDiskUse=True)

# Ensure indexing for authorSummary collection
if 'authorSummary' in db.collection_names():
    db.authorSummary.create_index("coauthors")
    db.authorSummary.create_index("years")
    db.authorSummary.create_index("papers")

# Create collection authorSummaryByYear
if 'authorSummaryByYear' not in db.collection_names():
    print("Collection authorSummaryByYear not found,
↪   creating...")
    db.NSR.aggregate([
        {"$project": {"_id": 1, "copyauthors": "$authors",
↪   "authors": 1, "year": 1}},
        {"$unwind": "$authors"},
        {"$unwind": "$copyauthors"},
        {"$group": {"_id": {"author": "$authors", "year":
↪   "$year"}, "coauthors": {
            "$addToSet": "$copyauthors"}, "papers": {"$addToSet":
↪   "$_id"}}},
        {"$out": "authorSummaryByYear"}
    ], allowDiskUse=True)

# Ensure indexing for authorSummaryByYear collection
if 'authorSummaryByYear' in db.collection_names():
    db.authorSummaryByYear.create_index("coauthors")
    db.authorSummaryByYear.create_index("papers")


# Generate authorSummary tsv for clustering
if not os.path.exists('author-cluster-input.tsv'):
    print("Aggregating authorSummary data...")
    authorSummary_pipeline = [
        {"$project": {"_id": 0, "author": "$_id", "numCoauthors": {"$size": "$coauthors"},
                      "numYears": {"$size": "$years"}, "numEntries": {"$size": "$papers"}}}
    ]
    results = db.authorSummary.aggregate(authorSummary_pipeline, allowDiskUse=True)
    with open('author-cluster-input.tsv', 'w', newline='') as tsvfile:
        print("Writing authorSummary data to file...")
        cluster_writer = csv.writer(tsvfile, delimiter='\t')
        cluster_writer.writerow(["author", "numCoauthors", "numYears", "numEntries"])
        for document in results:
            cluster_list = [document['author']]
            cluster_list.append(document['numCoauthors'])
            cluster_list.append(document['numYears'])
            cluster_list.append(document['numEntries'])
            cluster_writer.writerow(cluster_list)


# Generate authorSummaryByYear tsv for clustering
if not os.path.exists('author-cluster-entry-quartiles-input.tsv'):
    print("Aggregating authorSummaryByYear data...")
    authorSummaryByYear_pipeline = [
        {"$group": {"_id": "$_id.author", "yearData": { "$push":
            {"year": "$_id.year", "numCoauthors": {"$size": "$coauthors"}, "numEntries": {"$size": "$papers"}}}}},
        {"$project": {"author": "$_id", "yearData": 1, "numYears": {"$size": "$yearData"}}}
    ]
    results = db.authorSummaryByYear.aggregate(authorSummaryByYear_pipeline, allowDiskUse=True)
    with open('author-cluster-entry-quartiles-input.tsv', 'w', newline='') as tsvfile:
        print("Writing authorSummaryByYear data to file...")
        cluster_writer = csv.writer(tsvfile, delimiter='\t')
        cluster_writer.writerow(["author", "careerLength", "meanCoauthors",
                                 "numEntries", "numEntries033", "numEntries066", "numEntries100"])
        for document in results:
```

```python
            years = []
            entries = []
            coauthors = []
            for yearDatum in document['yearData']:
                years.append(yearDatum['year'])
                coauthors.append(yearDatum['numCoauthors'])
                entries.append(yearDatum['numEntries'])
            assert int(len(years)) == int(document['numYears']), "len(years) should be equal to numYears"
            if sum(entries) <= 10: continue
            years, entries = zip(*sorted(zip(years,entries), key=lambda x: x[0]))
            sumEntries = [entries[years.index(t)] if t in years else 0 for t in range(min(years), max(years)+1)]
            assert len(sumEntries) == max(years)-min(years)+1
            for i, entry in enumerate(sumEntries):
                if i >= 1:
                    sumEntries[i] = sumEntries[i] + sumEntries[i - 1]
            assert sumEntries[-1] == sum(entries), "sumEntries[-1] should be equal to sum(entries)"
            numEntries033 = percentile(sumEntries, 0.33)
            numEntries066 = percentile(sumEntries, 0.66) - numEntries033
            numEntries100 = sumEntries[-1] - numEntries066 - numEntries033
            cluster_list = [document['author']]
            cluster_list.append(max(years) - min(years)+1)
            cluster_list.append(statistics.mean(coauthors))
            cluster_list.append(sumEntries[-1])
            cluster_list.append(numEntries033 / sumEntries[-1])
            cluster_list.append(numEntries066 / sumEntries[-1])
            cluster_list.append(numEntries100 / sumEntries[-1])
            cluster_writer.writerow(cluster_list)

# Generate paper -> author transactions tsv
if not os.path.exists('transactions-papers-authors.tsv'):
    print("Generating papers -> authors transaction tsv files for association rule learning...")
    pipeline = [
        {"$match": {"authors": {"$exists": True}}},
        {"$project": {"_id": 0, "authors": "$authors"}}
    ]
    results = db.NSR.aggregate(pipeline, allowDiskUse=True)
    with open('transactions-papers-authors.tsv', 'w', newline='') as tsvfile:
        transaction_writer = csv.writer(tsvfile, delimiter='\t')
        for document in results:
            transaction_writer.writerow(document['authors'])

# Generate papers -> selectors transactions tsv
if not os.path.exists('transactions-papers-selectors.tsv'):
    print("Generating papers -> selectors transaction tsv files
↪   for association rule learning...")
    pipeline = [
        {"$match": {"selectors": {"$exists": True}}},
        {"$project": {"_id": 0, "selectors": 1}}
    ]
    results = db.NSR.aggregate(pipeline, allowDiskUse=True)
    with open('transactions-papers-selectors.tsv', 'w',
↪   newline='') as tsvfile:
        transaction_writer = csv.writer(tsvfile, delimiter='\t')
        for document in results:
            selector_list = [s['type'] + " " + s['value'] for s
↪   in document['selectors']]
            transaction_writer.writerow(selector_list)

# Generate selectors -> authors transactions tsv
if not os.path.exists('transactions-selectors-authors-set.tsv'):
    print("Generating selectors -> authors transaction tsv files
↪   for association rule learning...")
    pipeline = [
        {"$match": {"selectors": {"$exists": True}}},
        {"$unwind": "$selectors"},
        {"$unwind": "$authors"},
        {"$group": {"_id": "$selectors", "authors": {"$addToSet":
↪   "$authors"}}},
        {"$project": {"_id": 0, "authors": "$authors"}}
    ]
    results = db.NSR.aggregate(pipeline, allowDiskUse=True)
    with open('transactions-selectors-authors-set.tsv', 'w',
↪   newline='') as tsvfile:
        transaction_writer = csv.writer(tsvfile, delimiter='\t')
        for document in results:
            transaction_writer.writerow(document['authors'])
```

8.5 update-database.py

```python
import csv
import os.path
import pymongo
import subprocess
import sys
import argparse

# Parse command line input (filename of cluster memberships)
parser = argparse.ArgumentParser()
parser.add_argument('filename', type=argparse.FileType('r'))
args = parser.parse_args()
membership_document = args.filename

# Connect to the local Mongo server
try:
    client = pymongo.MongoClient('localhost', 27017,
    ↪ serverSelectionTimeoutMS=100)
    client.admin.command('ismaster')  # Test command to see if we
    ↪ can connect

    # If we can conenct, reset client to defaults
    print("Successfully connected to MongoDB...")
    client = pymongo.MongoClient('localhost', 27017)
except pymongo.errors.ConnectionFailure:
    sys.exit("ERROR: Could not connect to database, are you sure
    ↪ 'mongod' is running?")

# Check for 'masters' database
if 'masters' not in client.database_names():
    sys.exit("ERROR: Database 'masters' was not found. Inspect
    ↪ database and run 'prepare-data.py' if needed.")
else:
    print("Found database 'masters'...")
db = client['masters']


# Update collection authorSummary
if 'authorSummary' not in db.collection_names():
    sys.exit("ERROR: Collection 'authorSummary' was not found.
    ↪ Inspect database and run 'prepare-data.py' if needed.")
else:
    author_reader = csv.reader(membership_document,
    ↪ delimiter='\t')
    for author_line in author_reader:
        author = author_line[0]
        cluster_membership = author_line[1]
        db.authorSummary.update_one({"_id": author},{"$set":
 ↪  {"cluster": cluster_membership}})
```

8.6 calc-author-name-dist.py

```python
import csv
import jellyfish

def dist_calc(author_pair):
    dist = jellyfish.damerau_levenshtein_distance(author_pair[0],
↪   author_pair[1])
    if dist <= 3:
        author_pair.append(dist)
        return author_pair
    else:
        return False

with open('author-cluster-input.tsv', 'r', newline='') as
↪   tsvfile:
    reader = csv.DictReader(tsvfile, delimiter='\t', )
    authors = []
    for line in reader:
        authors.append(line['author'])

with open('author-name-ld-distance.tsv', 'w', newline='') as
↪   tsvfile:
    dist_writer = csv.writer(tsvfile, delimiter='\t')
    dist_writer.writerow(["author1", "author2", "ld_dist"])
    for i,first_author in enumerate(authors):
        for second_author in authors[i+1:]:
            dist_pair = dist_calc([first_author, second_author])
            if dist_pair:
                dist_writer.writerow(dist_pair)
```

8.7 calc-author-name-transform-pairs.py

```python
import argparse
import csv
from collections import defaultdict
import re

parser = argparse.ArgumentParser()
parser.add_argument('filename')
args = parser.parse_args()
author_list = args.filename

spacing = re.compile(r'\s*')
punc = re.compile(r'[^a-zA-Z0-9]')

names_lower = defaultdict(list)
names_nospace = defaultdict(list)
names_nopunc = defaultdict(list)

def build_transformation_dicts(inputfile):
    with open(inputfile, 'r', newline='') as tsvfile:
        reader = csv.DictReader(tsvfile, delimiter='\t')
        for line in reader:
            author = line['author']
            lower = author.lower()
            names_lower[lower].append(author)
            nospace = re.sub(spacing, '', author).lower()
            names_nospace[nospace].append(author)
            nopunc = re.sub(punc, '', author).lower()
            names_nopunc[nopunc].append(author)

def write_transformation_pairs(dictionary, filename):
    with open(filename, 'w', newline='') as tsvfile:
        pair_writer = csv.writer(tsvfile, delimiter='\t')
        for k in dictionary.keys():
            if len(dictionary[k]) >= 2:
                pair_writer.writerow(dictionary[k])

build_transformation_dicts(author_list)
write_transformation_pairs(names_lower, 'author-lower-pairs.tsv')
write_transformation_pairs(names_nospace,
 ↪  'author-nospace-pairs.tsv')
write_transformation_pairs(names_nopunc,
 ↪  'author-nopunc-pairs.tsv')
```

8.8 calc-cosine-sims.py

```
import pymongo
import sys
import csv
import functools
from collections import defaultdict
from gensim import corpora, models, similarities


# Connect to the local Mongo server
try:
    client = pymongo.MongoClient('localhost', 27017,
↪   serverSelectionTimeoutMS=100)
    client.admin.command('ismaster')  # Test command to see if we
↪   can connect

    # If we can conenct, reset client to defaults
    print("Successfully connected to MongoDB...")
    client = pymongo.MongoClient('localhost', 27017)
except pymongo.errors.ConnectionFailure:
    sys.exit("ERROR: Could not connect to database, are you sure
↪   'mongod' is running?")

# Create 'masters' database
if 'masters' not in client.database_names():
    sys.exit("ERROR: Database 'masters' was not found. Inspect
↪   database and run 'prepare-data.py' if needed.")
else:
    print("Found database 'masters'...")
db = client['masters']


# Get the selectors for all papers
selectors_pipeline = [
        {"$match": {"selectors": {"$exists": True}}},
        {"$project": {"_id":1, "selectors":1}}
]
results = db.NSR.aggregate(selectors_pipeline, allowDiskUse=True)

# Build the corpus
corpus = []
keynum_list = []
for document in results:
    vec = []
    for selector in document['selectors']:
```

```python
        if selector['value'] != "OTHER":
            #vec.append((selector['type'], selector['value']))
            vec.append(selector['type'] + " " + selector['value'])
    keynum_list.append(document['_id'])
    corpus.append(vec)

# Trim the corpus
frequency = defaultdict(int)
for selector_list in corpus:
    for token in selector_list:
        frequency[token] += 1
corpus = [[token for token in selector_list if frequency[token] > 1]
         for selector_list in corpus]

# Make a corpura dictionary
dictionary = corpora.Dictionary(corpus)
dictionary.save('selector-corpus.dict') # store the dictionary, for future reference

sparse_vectors = [dictionary.doc2bow(selector_list) for selector_list in corpus]
corpora.MmCorpus.serialize('selector-sparse-vectors.mm', sparse_vectors) # store to disk, for later use

# Initialize a transform
tfidf = models.TfidfModel(sparse_vectors)
index = similarities.Similarity('./', sparse_vectors, num_features=len(frequency))
index.num_best = 20

#with open('similar-papers-cosine-selectors.tsv', 'w', newline='') as tsvfile:
#    print("Writing similar papers data to file...")
#    sim_writer = csv.writer(tsvfile, delimiter='\t')
#    for i,paper in enumerate(corpus):
#        sims = index[tfidf[dictionary.doc2bow(corpus[i])]]
#        sim_list = [keynum_list[hisim[0]] for hisim in sims if hisim[1] > 0.65]
#        sim_list.insert(0, keynum_list[i])
#        sim_writer.writerow(sim_list)

for i,paper in enumerate(corpus):
    sims = index[tfidf[dictionary.doc2bow(corpus[i])]]
    sim_list = [{"paper": keynum_list[hisim[0]], "score": hisim[1]} for hisim in sims if hisim[1] > 0.65 and hisim[0] != i]
    db.simNSR.update_one({"_id": keynum_list[i]},{"$set": {"simPapers": sim_list}})
```

8.9 parse-arules-output.py

```
import re
import csv

with open('myoutput.tsv', 'r') as tsvfile:
    #Rule looks like: {"J.Pereira","L.Audouin","T.Enqvist"} =>
↪   {"P.Napolitani"}
    rules_reader = csv.reader(tsvfile, delimiter='\t')
    rules_iter = iter(rules_reader)
    next(rules_iter) #skip tsv header
    for rule in rules_iter:
        rule_no_brackets = re.sub(r'[{}]',r'',rule[1])
        rule_lhs,rule_rhs = re.split(r' => ',rule_no_brackets,
↪   maxsplit=1)
        rule_lhs = re.split(r',', rule_lhs)
        print(rule_lhs)
```

8.10 nsr_app.py

```python
from pymongo import MongoClient
from flask import Flask, jsonify, request, render_template
import networkx as nx
from networkx.readwrite import json_graph
from itertools import combinations
from collections import defaultdict
import re
import json
from bson import json_util
from bson.objectid import ObjectId
import csv
import numpy as np

def toJson(data):
    """
    Convert Mongo object(s) to JSON
    """
    return json.dumps(data, default=json_util.default)

# Read in static cluster info
cluster_info = defaultdict(int)
with open('../data/results/k3p3c0.tsv') as cluster_file:
    cluster_reader = csv.reader(cluster_file, delimiter='\t')
    for author_line in cluster_reader:
        cluster_info[author_line[0]] = author_line[1]

# Setup the connection to MongoDB and get the NSR collection
client = MongoClient('localhost', 27017)
db = client.masters
nsr = db.NSR

# Create our instance of the Flask class
app = Flask(__name__)

# Enable CORS
# https://gist.github.com/blixt/54d0a8bf9f64ce2ec6b8
# This should be investigated for security concerns
def add_cors_headers(response):
    response.headers['Access-Control-Allow-Origin'] = '*'
    if request.method == 'OPTIONS':
        response.headers['Access-Control-Allow-Methods'] = 'DELETE, GET, POST, PUT'
        headers = request.headers.get('Access-Control-Request-Headers')
        if headers:
            response.headers['Access-Control-Allow-Headers'] = headers
    return response
app.after_request(add_cors_headers)

# Root site route
@app.route('/')
def hello_world():
    """
    Serves the main web interface.
    """
    return render_template('index.html')

# Route for find_one(year)
@app.route('/api/year/<int:year_id>')
def find_year(year_id):
    """
    This code is not used in the main application.
    It is kept as a small example of a possible API endpoint.
    """
    first_doc = nsr.find_one({"year": year_id})
    title = first_doc['title']
    return "The first title in {year} is: {title}".format(year=year_id, title=title)

# Main search route
@app.route('/api/search')
def parse_search():
    """
    We apply a series of RegEx to the user search query and use tge matches to
    build a aggregation pipeline to pass to the NSR collection in MongoDB.
    """
    # Get the query parameter 'input' and setup an empty pipeline list
    search = request.args['input']
    print("NSR> recieved search: " + search)
    pipeline = []
    simpapers_pipeline = []
    options = {}

    # Try to match years (only supports four digit years)
    year_list = re.findall(r"(?<![:=_])([12][0-9]{3})+", search)
    if len(year_list) == 1:
        pipeline.append({"$match": {"year": int(year_list[0])}})
    if len(year_list) == 2:
        year_start = int(min(year_list))
        year_end = int(max(year_list))
        pipeline.append({"$match": {"year": {"$gte": year_start, "$lte": year_end}}})
    if len(year_list)  > 2:
        pipeline.append({"$match": {"year": {"$in": year_list}}})

    # Try to match author names (room for improvement)
    author_tuples = re.findall(r"(?<![:=_])(([a-zA-Z]\.){1,3}[a-zA-Z-]+)+", search)
    if len(author_tuples) >= 1:
        author_list = [author[0] for author in author_tuples]
        pipeline.append({"$match": {"authors": {"$in": author_list}}})
        # Copy the pipeline for use with simpapers query
        simpapers_pipeline = list(pipeline)

    # Try an match nuclides written like: 11li 12C 238U
    nuclide_tuples = re.findall(r"(?<![:=_])([0-9]{1,3}[a-zA-Z]{1,3})+", search)
    if len(nuclide_tuples) >= 1:
        nuclide_list = [nuclide.upper() for nuclide in nuclide_tuples]
        pipeline.append({"$match": {"selectors.value": {"$in": nuclide_list}}})

    # Format data with a projection and then perform the aggregation
    # The pipeline before projection is a good candidate for building a cache
    pipeline.append({"$project":
        {"_id": 1, "year": 1, "authors": 1, "type": 1, "selectors": "$selectors.value", "title": 1}})
    results = nsr.aggregate(pipeline)


    # Similar papers
    simpaper_entries = []
    if len(simpapers_pipeline) > 0:
        simpapers_pipeline.extend([
            {"$unwind": "$simPapers"},
            {"$group": {"_id": "$simPapers.paper", "score": {"$avg": "$simPapers.score"}}},
```

```python
            {"$sort": {"score": -1}},
            {"$limit": 100}
        ])
        recommended_papers = db.simNSR.aggregate(simpapers_pipeline)

        # save the score for each recommended paper
        scores = dict()
        for paper in recommended_papers:
            scores[paper['_id']] = paper['score']

        # get the full documents for each paper
        simpaper_results = nsr.aggregate([
            {"$match": {"_id": {"$in": [x for x in scores.keys()]}}},
            {"$project": {"_id": 1, "year": 1, "authors": 1, "type": 1,
                "selectors": "$selectors.value", "title": 1}}])
        simpaper_entries = []
        for simpaper in simpaper_results:
            simpaper['score'] = "Score: {:6.4f}".format(scores[simpaper['_id']])
            simpaper_entries.append(simpaper)


    # Iterate over MongoDB cursor and update defaultdict values in _dict vars
    # This can throw IndexErrors which I am not catching
    types_dict = defaultdict(lambda: defaultdict(int)) #2 level deep defaultdict with in int
    years_dict = defaultdict(int)
    nsr_entries = []
    G = nx.Graph()
    author_nodes = set()
    for index, nsr_entry in enumerate(results):
        nsr_year = int(nsr_entry['year'])
        nsr_type = nsr_entry['type'] if 'type' in nsr_entry else 'UNKNOWN'
        years_dict[nsr_year] += 1
        types_dict[nsr_year][nsr_type] += 1

        # Limit nsr_entries and graph to only 1000 results
        if index <= 1000:
            nsr_entries.append(nsr_entry)
            if 'authors' in nsr_entry:
                author_nodes.update(nsr_entry['authors'])
            for i in combinations(nsr_entry['authors'], 2):
                G.add_edge(i[0], i[1])

    # Layout graph
    G.add_nodes_from(author_nodes)
    dist_scale = pow(G.number_of_nodes(), -0.3)
    # Fix Numpy random seed to generate reproducible graphs
    np.random.seed(0)
    positions=nx.spring_layout(G, k=dist_scale, iterations=75)
    #positions=nx.spectral_layout(G)

    # Split network-graph into components
    if 'topnetwork' in options and int(options['topnetwork']) !=
↪   0:
        graphs = list(nx.connected_component_subgraphs(G))
        graphs.sort(key = lambda x: -x.number_of_edges())
        top_num = int(options['topnetwork'])
        network_data =
↪   json_graph.node_link_data(graphs[top_num-1])
    else:
        network_data = json_graph.node_link_data(G)

    # Add positions to network_data
    for n in network_data['nodes']:
        n['x'] = int(positions[n['id']][0]*400+400)
        n['y'] = int(positions[n['id']][1]*400+400)
        n['k'] = cluster_info[n['id']]

    year_start = min(years_dict.keys())
    year_end = max(years_dict.keys())
    nsr_types = ['JOUR', 'CONF', 'REPT', 'PC', 'BOOK', 'THESIS',
↪   'PREPRINT', 'UNKNOWN']

    # Transform year data to NVD3 format for multibar chart: [{x:
↪   1989, y: 10}, {x: 1999, y: 20}, ...]
    # Additionally we build x,y dicts for years that do not
↪   appear in the results to avoid sparse data
    year_json = [{'x': year, 'y': years_dict[year]} for year in
↪   range(year_start, year_end + 1)]

    # Transforms year and type data to NVD3 format for multibar
↪   chart where each type is given a key
    # We use the nsr_types list to ensure all types are
↪   represented even in sparse data
    types_json = [{'key': _type, 'values': [ {'x': year, 'y':
↪   int(types_dict[year][_type])} for year in
```

```
        range(year_start, year_end + 1) ]} for _type in
↪   nsr_types]

    # Convert everything to JSON and ship it to the client
    return toJson({'years': year_json, 'types': types_json,
↪   'entries': nsr_entries, 'network': network_data, 'simpapers':
↪   simpaper_entries})


# If executed directly from python interpreter, run local server
if __name__ == '__main__':
    app.run(debug=True)
```

8.11 main.js

```
// Enable debugging output
var verbose = true;

// Searches made with the navbar search form
document.getElementById("omnisearch").addEventListener("search",
↪  omniSearch);

function omniSearch(ev){

    if(verbose){console.log("Running omniSearch()...");};

    // Break the search value into key:value pairs
    var search = document.getElementById("omnisearch").value;
    var queryItems = search.split(/ (?=\w+:)/);
    var command = queryItems[0].split(":")[0];
    var commandParam = queryItems[0].split(":")[1]

    //Create options object from queryItems
    var options = {};
    for (i=1; i<queryItems.length; i++){
        queryItem = queryItems[i].split(":")
        options[queryItem[0]] = JSON.parse(queryItem[1]);
    }
    if(!options.topnetwork) options.topnetwork = 0;

    // Erase the current #charts output
    document.getElementById("charts").innerHTML = '';

    // Handle the passed search command
    switch (command){

        default:
            // Send entire input to /api/search
            d3.xhr("/api/search?input="+search)
            .get(function(error, data){
                response = JSON.parse(data.response)
                listEntries(response.entries)
            })
            break;

    }//end of switch on search command

    ev.preventDefault();
};//end of search
```

8.12 force-graph.js

```
/* Define the main worker or execution function */
// Possible options for the force graph are:
// radius [5]
// links [30]
// charge [-120]
// friction [0.8]
// gravity [0.1]
// labels [off]
// drag [off]

function forceDirectedGraph(error, nodes, links, options) {

    d3.select("#charts").remove()
    var svg = d3.select("div#contentContainer")
       .append("div")
       .classed("svg-container", true) //container class to make
 ↪  it responsive
       .attr("id", "charts")
       .append("svg")
       //responsive SVG needs these 2 attributes and no width and
 ↪  height attr
       //.attr("preserveAspectRatio", "xMinYMin meet")
       .attr("viewBox", "0 0 1000 1000")
       //class to make it responsive
       .classed("svg-content-responsive", true);

    var container = svg.append("g")

    // Optional radius customization or set default
    var radius = options.radius || 5;
    // Set the color scale we want to use
    var color = d3.scale.category10();

    // Draw the edges/links between the nodes
    var edges = container.selectAll("line")
        .data(links)
        .enter()
        .append("line")
        .style("stroke", "#666")
        .style("stroke-opacity", ".2")
        .style("stroke-width", 1)
        .attr("x1", function(d) { return nodes[d.source].x; })
        .attr("y1", function(d) { return nodes[d.source].y; })
        .attr("x2", function(d) { return nodes[d.target].x; })
        .attr("y2", function(d) { return nodes[d.target].y; });

    // Options node labels
    if (options.labels) {
        var texts = container.selectAll("text")
            .data(nodes)
            .enter()
            .append("text")
            .attr("fill", "black")
            .attr("font-family", "sans-serif")
            .attr("font-size", "12px")
            .attr("x", function(d) { return d.x + 5; })
            .attr("y", function(d) { return d.y + 5; })
            .text(function(d) { return d.id; });
    }

    // Draw the nodes themselves
    var nodes = container.selectAll("circle")
        .data(nodes)
        .enter()
        .append("circle")
        .attr("r", radius)
        .style("fill", function(d,i) { return color(d.k); })
        .style("stroke", "#fff")
        .style("stroke-width", "1px")
        .attr("cx", function(d) { return d.x; })
        .attr("cy", function(d) { return d.y; })

}; // End forceDirectedGraph worker func

function stackedBar(data) {
    d3.select("#charts").remove()
    d3.select("#contentContainer").append("div").attr("id",
 ↪  "charts").append("svg")
        .attr("width", 600).attr("height", 400)
    nv.addGraph(function() {
        var chart = nv.models.multiBarChart().stacked(true);

        chart.xAxis
            .tickFormat(d3.format(''));

        chart.yAxis
            .tickFormat(d3.format(''));

        d3.select('#charts svg')
            .datum(data)
            .call(chart);

        nv.utils.windowResize(chart.update);
        chart.update()

        return chart;
    });
}

function listEntries(data) {
    d3.select("#charts").remove()
    d3.select("#contentContainer").append("div").attr("id", "charts")
    data.forEach(function (nsr){
        d3.select("#charts").append("div").attr("class", "nsrEntry").attr("id", "id" + nsr._id)
        var entry = d3.select("#id" + nsr._id)
        entry.append("span").attr("class", "year").text(nsr.year);

        var title = typeof nsr.title === "undefined" ? "NA" : nsr.title
        entry.append("span").attr("class", "title").text(title);

        var score = typeof nsr.score === "undefined" ? "" : nsr.score
        entry.append("span").attr("class", "score").text(score);

        var authors = typeof nsr.authors === "undefined" ? "NA" : nsr.authors.join(", ")
        entry.append("p").attr("class", "authors").text(authors);

        var selectors = typeof nsr.selectors === "undefined" ? "NA" : nsr.selectors.join(", ")
        entry.append("p").attr("class", "selectors").text("Selectors: " + selectors);
    })

}
```

Bibliography

[1] A. H. Bécquerel, C. R. Acad. Sci. Paris **122**, 501 (1896).

[2] L. Kurgan and P. Musilek, The Knowledge Engineering Review **21**, 1 (2006).

[3] B. Pritychenko, E. Běták, M. A. Kellett, B. Singh, and J. Totans, Nuclear Instruments and Methods in Physics Research A **640**, 213 (2011).

[4] B. Pritychenko, *Private Communication* (2015).

[5] D. F. Winchell, Science 3 (2007).

[6] Wikipedia, *Digital Object Identifier — Wikipedia, the Free Encyclopedia* (2015).

[7] E. F. Codd, Commun. ACM **13**, 377 (1970).

[8] PyMongo, *PyMongo 3.0.3 Documentation* (2015).

[9] Wikipedia, *MongoDB — Wikipedia, the Free Encyclopedia* (2015).

[10] MongoDB, *The MongoDB 3.0 Manual* (2015).

[11] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow, CoRR **abs/1111.4503**, (2011).

[12] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, Journal of Statistical Mechanics: Theory and Experiment **10**, 8 (2008).

[13] C. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval* (Cambridge University Press, 2009).

[14] G. Salton, A. Wong, and C. S. Yang, Commun. ACM **18**, 613 (1975).

[15] D. Dubin, Library Trends 2004 (n.d.).

[16] S. K. M. Wong and V. V. Raghavan, in *Research and Development in*

Information Retrieval (1984), pp. 167–185.

[17] K. Sparck Jones, Journal of Documentation **28**, 11 (1972).

[18] Z. E. Xu, M. Chen, K. Q. Weinberger, and F. Sha, CoRR **abs/1301.6770**, (2013).

[19] J. B. Lovins, (1968).

[20] M. Porter, *The Lovins Stemming Algorithm* (2015).

[21] M. Porter, Program **40**, 211 (2006).

[22] P. Willett, Program **40**, 219 (2006).

[23] A. Huang, in *Proceedings of the Sixth New Zealand Computer Science Research Student Conference (NZCSRSC2008), Christchurch, New Zealand* (2008), pp. 49–56.

[24] Wikipedia, *Vector Space Model — Wikipedia, the Free Encyclopedia* (2015).

[25] B. Pritychenko, CoRR **abs/1411.1899**, (2014).

[26] G. Navarro, ACM Comput. Surv. **33**, 31 (2001).

[27] V. I. Levenshtein, Soviet Physics Doklady **10**, 707 (1966).

[28] D. Blei, *Probabilistic Topic Models* (2012).

[29] D. Blei and J. Lafferty, *Topic Models* (2009).

[30] D. M. Blei, A. Y. Ng, and M. I. Jordan, The Journal of Machine Learning Research **3**, 993 (2003).

[31] P. Pinoli, D. Chicco, and M. Masseroli, Computational Intelligence in Bioinformatics and Computational Biology, 2014 IEEE Conference on 1 (2014).

[32] M. Nikolić, Intelligent Data Analysis **40**, (2012).

[33] C. C. Aggarwal, editor, *Data Classification: Algorithms and Applications*

(CRC Press, 2014).

[34] M. Hahsler, C. Buchta, B. Gruen, and K. Hornik, *Arules: Mining Association Rules and Frequent Itemsets* (2015).

[35] M. Hahsler, B. Gruen, and K. Hornik, Journal of Statistical Software **14**, 1 (2005).

[36] R. Agrawal, R. Srikant, and others, in *Proc. 20th Int. Conf. Very Large Data Bases, VLDB* (1994), pp. 487–499.

[37] J. MacQueen and others, Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability **1**, 281 (1967).

[38] A. K. Jain, Pattern Recognition Letters **31**, 651 (2010).

[39] A. K. Jain, M. N. Murty, and P. J. Flynn, ACM Comput. Surv. **31**, 264 (1999).

[40] R. Tibshirani, G. Walther, and T. Hastie, Journal of the Royal Statistical Society: Series B (Statistical Methodology) **63**, 411 (2001).

[41] D. L. Davies and D. W. Bouldin, Pattern Analysis and Machine Intelligence, IEEE Transactions on 224 (1979).

[42] Wikipedia, *Davies–Bouldin Index — Wikipedia, the Free Encyclopedia* (2015).

[43] T. Calinski and J. Harabasz, Communications in Statistics-Theory and Methods **3**, 1 (1974).

[44] ttnphns, *What Is an Acceptable Value of the Calinski & Harabasz (CH) Criterion?* (2015).

[45] M. Walesiak, A. Dudek, and, *ClusterSim: Searching for Optimal Clustering Procedure for a Data Set* (n.d.).