SNP based literature and data retrieval

UNIVERSITY *of the* WESTERN CAPE

Werner Pieter Veldsman

A thesis submitted in partial fulfillment of the requirements
for the degree of Magister Scientiae at the South African National Bioinformatics Institute
in the Faculty of Natural Sciences, University of the Western Cape

Supervisor: Prof A. Christoffels

March 2016

As ek eendag in die hemel kom,

Sal God die Vader

Nie omgee nie

as ek my pa eerste groet,

soos altyd

Met 'n “Hei Stofjas!”

Want God weet

Ek het ook na hom verlang

KEYWORDS

API (Application Programming Interface)

Bioinformatics

Data mining

F/oss (Free and open source software)

Literature sourcing

PMC-OAI (PubMed Central Open Access Initiative)

Search engine

SNP (Single Nucleotide Polymorphism)

VCF (Variant Call Format)

Web application

ABSTRACT

SNP based literature and data retrieval

W.P. Veldsman

Magister Scientiae at the South African National Bioinformatics Institute in the Faculty of Natural Sciences, University of the Western Cape

In this thesis, the development of a bioinformatics software package is motivated, planned and implemented as a web application (http://sniphunter.sanbi.ac.za) with an application programming interface (API). The purpose is to allow scientists searching for relevant literature to query a database using reference single nucleotide polymorphism (SNP) identifiers and potential keywords assigned to scientific literature by the authors. Multiple queries can be simultaneously launched using either the web interface or the API. In addition, a variant call format (VCF) file parser was developed and packaged with the application to allow users to upload, extract and write information from VCF files to a file format that can be interpreted by the novel search engine created during this project. The parsing feature is seamlessly integrated with the web application's user interface, meaning there is no expectation on the user to learn a scripting language.

This multi-faceted software system, called SNiPhunter, envisions saving researchers time during life sciences literature procurement, by suggesting articles based on the amount of times a reference SNP identifier has been mentioned in an article. This will allow the user to make a quantitative estimate as to the relevance of the article. A second novel feature is the inclusion of the email address of a correspondence author in the results returned to the user, which promotes communication between scientists. Moreover, links to external functional information are provided to allow researchers to examine annotations, associated with their reference SNP identifier of interest, in the National Center for Biotechnology Information database. Standard information such as digital object identifiers and publishing dates, that are typically provided by other search engines, are also included in the results returned to the user.

March 2016

DECLARATION

I declare that *SNP based literature and data retrieval* is my own work, that it has not been submitted before for any degree or examination in any other university, and that all the sources I have used or quoted have been indicated and acknowledged as complete references.

_____________________ March 2016

Werner P. Veldsman

ACKNOWLEDGMENTS

This project was financially supported by the South African National Research Foundation.

I would like to thank Prof Alan Christoffels for his suggestions and continued guidance in keeping this project relevant to the broader goals of the Southern African Human Genome Programme. I'm also grateful to massive open online course (MOOC) providers Coursera and EdX for publishing free of charge audiovisual courses on software design.

The following list recognizes the contribution of third party software in making this project possible (with copyright owners named in parenthesis). All these packages are available on Github and are listed in the chronological order that they were used during the development of SNiPhunter:

- jQuery (jQuery Foundation)
- Bootstrap (Twitter, Inc.)
- JSON-server (Typicode)
- Awesomplete (Lea Verou)
- Forever (Charlie Robins & the contributers)
- Forever-service (Zapty Inc, Arvind Agarwal & the contributors)
- Highlight (Ivan Sagalaev)
- Awk [a wrapper of WebAwk] (Turing)
- Intercept-stdout (Steve Farthing)
- Multer (Hage Yaapa)

Some minor software contributions, that are not listed above, came in the form of snippets from programming websites such as Alec Jacobsen's blog and Fivera's CSS examples.

CONTENTS

INTRODUCTION

The volume of information from next-generation sequencing (NGS) is increasing at an exponential rate, and so is the amount of information available downstream in the bioinformatics analysis pipeline. To make efficient use of the data becoming available to scientists and the general public, databases have to be integrated to improve user experience (UX), by being readily accessible to multiple controller frameworks employing different standards and protocols. A conservative estimate, that excluded non peer reviewed databases and commercial databases, put the number of human related databases at 1550+ as at 2014 (Zou *et al*., 2015). These data sources are heterogeneous in content and globally distributed, which necessitates providing application programming interfaces (APIs) in addition to user interfaces (UIs).

The Southern African Human Genome Programme (SAHGP) is expected to produce South Africa's first bulk NGS data in the near future. Researchers will have access to this data in formats representing progressive stages of analysis. One of the final stages of NGS data processing is encapsulated in variant call format (VCF) files. The information on genetic variation contained within VCF files will assist researchers not directly involved with the SAHGP, in analyzing human specific experimental data, and in the generation of hypotheses leading to research outputs that could benefit the population of Southern Africa.

Conducting research using unstructured data is a time consuming process. As mentioned above, genome sequencing data is frequently processed and captured in VCF files. The content of files that use this format is then relied upon by researchers to, for example, construct queries for initiating academic literature resourcing using search engines. However, VCF files are not presented in a user friendly format. The parsing of their content using software libraries, such as VCFtools (a Perl library) and PyVCF (a Python library), is frequently required to extract useful information such as variant IDs that could serve as search query keywords. Moreover, literature retrieval search engines seem to (i) not provide an option to return results based on the occurrence frequency of a keyword, and (ii) not provide file upload facilities for bulk querying using keyword lists.

To address these shortcomings, the research question this project seeks to answer is: can data contained within VCF files, be integrated with the processes of keyword generation and search engine querying, to create a search engine that returns results with an empirical indication of the relevance of a result?

The expected outcome for this project is: to design and implement a concept search engine in answer to the research question, and to address the observed shortcomings in current academic search engines, with an emphasis on user experience (UX). A cornerstone of the software design approach is to use free and open source software (F/oss) in combination with open access scientific literature. In addition to this, the development of this proof-of-concept search engine steers clear of traditional database design software relying on structured query language (SQL) and the PHP HyperText Preprocessor, by using a NoSQL database supported by JavaScript and the Node interpreter. The purpose is to illustrate scientific software design using low-cost, untraditional alternatives.

A review of the literature will follow this introduction. Then the developmental approach of the project will be explained in detail. Next, a set of results will be presented and discussed. This thesis will end with a conclusion, a set of references and an addendum containing source code to supplement the methods and materials section.

LITERATURE REVIEW

Historical perspective

Early biological databases were created before the advent of the world wide web. That was in a world where biological data was shared by means of snail mailing data storage devices, such as punched paper cards or magnetic tapes, to scientific collaborators. Email and mass production of personal computers were yet to arrive on the scene. The first citation of digital data storage can be traced back to 1945 when the *memex* portmanteau was coined by Vannevar Bush using the words *memory* and *index* (Bush, 1945). This was done to give a name to a concept that envisioned people storing their books, communication and other records in a mechanized and easily retrievable manner in order to enhance human memory. Twenty years later the Cambridge Structural Database (CSD) started its transition from printed circulation to digital circulation (Attwood *et al.*, 2011). The CSD is recognized as one of the first scientific databases and as inspiration for the formation of the more widely known Protein Data Bank (PDB). The use of computers in the field of biology eventually led to the formation of the word *bioinformatics* (also known as computational biology or genomic data science) in the late 1970's. This marked the emergence of computer aided biological studies as a discipline in its own right. Shortly afterwards, in the early 1980's, scientists began to realize that the rate limiting step in nucleic acid sequencing was shifting from data acquisition to data management due to the emergence of faster sequencing technologies (Gingeras and Roberts, 1980). Collecting large amounts of scientific data and designing efficient search algorithms, were no longer enough. Information resource interoperability became a central concern of bioinformatics during the mid 1980's, when the volume of information being generated became more than the databases of the time could handle (Robbins, 1996). The problems faced during biological data management has remained an area of concern since then. In the late 2000's, the reason for this lack of progress in integration was ascribed to a lack of standardization and uncoordinated bioinformatic research, rather than to the volume of information (Goble & Stevens, 2008). Currently, semantic web technology is being investigated as an integration architecture that would allow for the substitution of links between data documents with links between the underlying data, thereby decreasing search times and allowing abstraction of information (Machado *et al*., 2015). Abstraction is vital for the application of *black box medicine* in future clinical settings, that is, Decision Support Systems (DSS), that will match clinical data with broadly distributed and heterogeneous knowledge bases (Price, 2015), without requiring the user to have technical expertise in database querying.

Literature review methodology

Point-of-view

Given the research question this project seeks to answer, and a trend towards using semantic web technology, the current state of the literature will be critically analyzed from the standpoint that integration of biological databases with structured and unstructured data, is necessary. The following literature review postulates that a gap exists in the process of biological literature retrieval. It is argued that this gap could be addressed by designing a tertiary data artifact with search engine functionality, that integrates relevant information from *open access* scientific literature, with parsed information from next generation sequencing data contained within VCF files.

Choice of academic search engine

In a comparison of Google Scholar (GS) with eleven other bibliographical databases, Walters (2009) found that GS outperformed its rivals by returning 41% of a set of preselected relevant records when all search results were taken into consideration. Although GS was not the top ranking academic search engine, when examining less than 75 results using the same criteria, it should be noted that GS has a much larger indexed literature set. On average GS returned a total of 20400 records per search, while the second most records returned by a competitor averaged at 311 records per search. A more specific examination of biomedical literature retrieval, compared GS to three other major search engines: Pubmed, Scopus and Web of Science (WoS) (Falagas *et al*., 2008). These authors concluded that Pubmed remains an important resource for clinicians, but that the volume of information available through GS made it a better choice when searching for less known papers. However, they did point out that citation analysis using GS fell short of their expectations, and a later study by Ortega and Aguillo (2014) determined that GS search results tended to be more biased towards computing and information science. Given the scope of this project, this bias is not a cause for concern. Ortega and Aguillo (2014) concluded that both GS and Microsoft Academic Search (MAS) should be used in conjunction with other citation indexes due to GS and MAS exhibiting technical limitations such as duplications and manipulations of results and citations. Their conclusion might have been too harsh given that Walters (2009) had already found GS to be a reliable indexer of relevant literature five years earlier. Moreover, paid access literature search engines, such as Scopus and WoS, could be overtaken by free search engines such as GS and Pubmed due to an increase in awareness of Open Access (OA) literature among researchers. Nevertheless, literature retrieved for the purposes of this study was sourced using both GS and Pubmed. Indeed, a subsection of this project relied on material acquired exclusively from Pubmed Central for the creation of a local database. One specific example of a Pubmed feature not available to GS users is the Medical Subject Headings (MeSH) term selector, which allows for filtering search results by selecting manually curated hierarchical keywords that are tagged to relevant articles.

Citation and rank analysis

Pubmed and GS have different approaches in terms of citation analysis and ranking of search results. These two properties have an effect on how article quality is perceived by the reader. Pubmed does not reflect citation counts in its search results, but instead lists articles in reverse chronological order by default (Falagas *et al*., 2008), while GS includes citation counts in its search results, but does not make its ranking criteria known to the public. GS offers citation analysis using the h-index formula. GS is considered one of the two major citation indexes, the other being the Thomson Reuters Impact Factor associated with the Institute of Scientific Information and its WoS search engine. The h-index allows for both publishers and authors to be ranked in terms of their productivity and apparent research output quality. Using the citation score of a publisher as an indication of the average quality of an article in a journal could be appropriate, but caution should be exercised in using this metric for determining material relevant to a literature review. This is due to a well known skewing in individual journal quality observed within publishing houses (Garfield, 2006). Therefore, individual article citation counts were only used as a general indication of quality for GS results during this literature review. The preference for a “more recent literature first” approach as seen in Pubmed's rankings did not affect the selection of material for this review significantly, although some articles were selected based on their date of authoring, especially whilst reviewing trends in technologies relevant to this project.

Keyword selection

While conducting searches using academic search engines, care should be taken during keyword submission to ensure that results will be efficient and reproducible. When entering search terms in GS and Pubmed, the following guidelines should be kept in mind:

- Boolean operators must be in capital letters (NOT, AND, OR). Their order of execution is from left to right in the search string, but the order can be predetermined on Pubmed by using parentheses.

- Search terms are case insensitive. It is therefore good practice to enter search terms using lowercase to distinguish them from boolean operators.

- The “AND” operator is implied and can be replaced by prefixing a plus sign to the term on GS or an ampersand to the term on Pubmed.

- The tilde sign (~) can be placed in front of a keyword to include synonyms in the search using GS. With Pubmed synonyms are automatically searched for when a term is suffixed with the MeSH operator “[mh]”.

- One should avoid using adverbs, conjunctions and prepositions. These words, also called stop-words, are ignored by search engines unless they are preceded by a boolean operator.

- Parentheses should be placed around phrases to search for keywords in their exact order while using GS. In contrast, Pubmed's search engine automatically searches for phrases, and researchers are advised to first attempt searching for phrases without using parentheses.

- Searches for plural word forms have to be explicitly declared in GS by adding the plural word form. In Pubmed, word stems are automatically included in searches when using MeSH terms.

- Precede a keyword with “title:” when using GS to request that all results contain the keyword in the article's title. Follow the search term with “[ti]” to accomplish the same in Pubmed.

In this literature review, primary keyword terms were identified by deconstruction of informative sentences in scientific articles acquired from publications such as the Bioinformatics journal (ISO4 abbr. *Bioinformatics)*. Secondary keyword terms were generated in a similar manner by using articles obtained from search results derived from the primary keyword terms. The following sentence will be used to demonstrate keyword construction:

A user that has a patient’s <u>informed consent</u> to analyze the <u>clinical data</u> may upload <u>sequence variants</u> to <u>GeneTalk</u> in variant call format (<u>VCF</u>)
(Kamphans and Krawitz, 2012)

The three phrases and two individual keywords underlined in this sentence are possible choices for inclusion as search terms. However, additional keywords might be necessary to avoid ambiguity. For example, the keyword *vcf* should be followed by the keyword *bioinformatics* to retrieve results relevant to biological data instead of electronic business cards (vCards) that are coincidently stored in files with the extension *vcf*. Alternatively, the acronym can be replaced by the complete phrase, *variant call format*.

Scope and structure of the review

Scope

Although this survey of the literature has properties of a systematic review, the selection of relevant material was not trimmed until a manageable volume of material remained. Instead, an iterative approach was followed that consisted of:

1. keyword generation
2. search engine submission

3. intuitive selection based on title, blurb, citation count, and publication date

The process was repeated until an argument could be sufficiently constructed from identification of:

- key trends and main theories
- proponents and opponents of themes
- examples from previous research and results
- all topics relevant to the project's implementation

Structure

The main body of the review that follows, will begin with a general discussion of major theories and models in database administration and biological data management. Critical analysis of topics and trends will be grouped together by theme. Benefits and drawbacks will be highlighted under each topic or trend, and where relevant, followed by subdivision and clustering of author opinions into proponents and critics. Finally, a conclusion will follow that reiterates support for this project to proceed.

Biological ontologies

Defining ontology

The science of ontology attempts to reconcile conflicting opinion regarding the difference between the abstract and the concrete by studying existence. The paradigm of ontology states that the problem of ambiguity in naming conventions has in its nature uncertainty brought about by conflicting schools of though regarding whether materialism or idealism best describes nature. However, the inherent nature of conflicting opinion already becomes apparent when attempting to define ontology. There are a multitude of definitions for this field of study with the following three equally correct definitions being put forward as applicable to biological database design:

An ontology is a concrete form of a conceptualization of a community's knowledge.

(Stevens *et al*., 2000)

A precise explanation of one's terms and reasoning in some subject area, which can allow computers to help, is called an ontology.

(The World Wide Web Consortium, 2015)

An explicit, formal representation of concepts and relationships among them within a particular domain that expresses human knowledge in machine readable form.

(Martone, 2012)

A simple explanation for what an ontology is in the domain of bioinformatics can be derived from the above definitions: it's a concept dictionary that can be used by both humans and machines. Moreover, an ontology is composed of a vocabulary of words together with a specification of their meaning, and an ontology has to be encoded using a knowledge representation (KR) language so that the ontology can serve as a data template. The semantic clarity that an ontology provides through KR encoding has to be balanced with available time and resources because creating ontologies is a time consuming process (Martone, 2012). The following diagram (Figure 1) illustrates a simple biological ontology using the subject of this project, deoxyribonucleic acid (DNA), as an example:

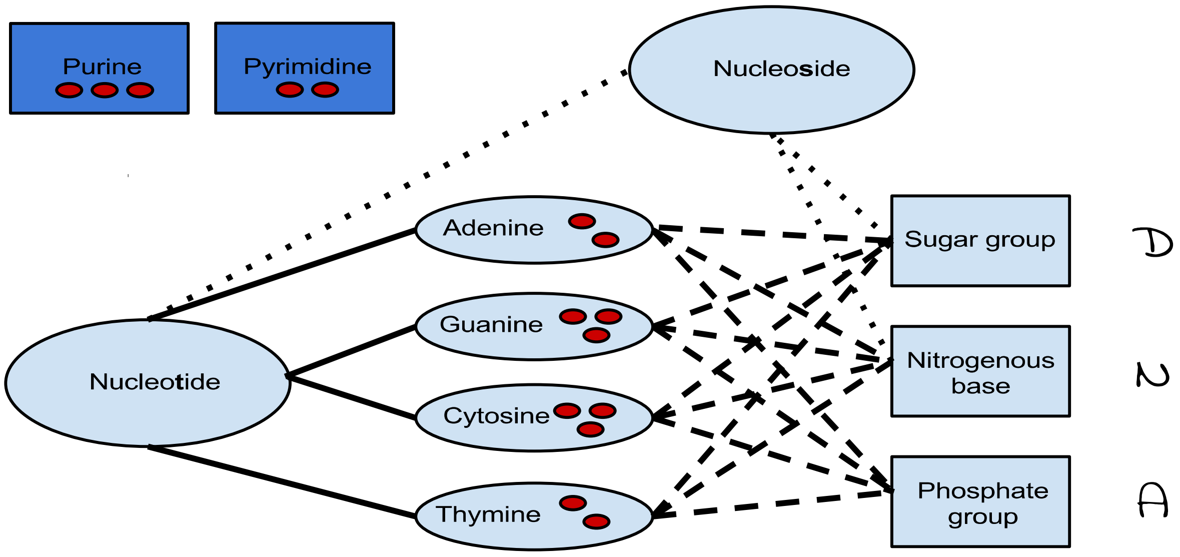


Figure 1: An ontology of the molecular structure of nucleotides

In the ontological diagram on the previous page the difference between nucleotides and nucleosides are illustrated using solid lines to indicate sub-classes (is_a relationships), while the non-solid lines refer to compositional relationships (has_a relationships). This exercise in deconstruction assigns compositional members to the class nucleosides (dotted lines) and the class nucleotides (striped lines), by using three biochemical functional groups that can form part of either of these two types of organic molecules. The red dots (pairs and triplets) in some of the classes indicate the amount of hydrogen bonds that each member of the class have that can engage in non-covalent bonding. This exemplifies the value of having automated reasoners since if a relationship has not been explicitly declared such as illustrated with the lack of connecting lines for the purine and pyrimidine classes (e.g. adenine is a member of the class pyrimidine), the relationship can be inferred by automatic reasoning based on the cardinality of the hydrogen bonding property. For example, if a cardinality rule states that all members of the class purine must have three and only three hydrogen bonds capable of forming non-covalent bonds in the context of DNA strand formation, then a computer will be able to determine that guanine and cytosine must be purines, while thymine and adenine cannot be purines. Automated reasoners are essential in the design of larger ontologies that may contain thousands of classes and will be discussed next.

Encoding an ontology

Ontologies can be prone to incongruences such as classes that cannot be instantiated or conflicting specifications. For example, imagine that a relationship between two entities has been declared as functional (being a single valued property). If a third entity is assigned the same relationship that exists between the first and second entity, the implication is that the second and third entities are the same entity. However, if the second and third entity have been explicitly declared as distinct entities, an inconsistency arises. Automated reasoners can be used to avoid these type of idiosyncratic artifacts as well as to infer relationships that were not manually declared. The specification of an ontology is often implemented with descriptive logic (DL), although some reasoners use a more reduced propositional logic (PL). Fact++ (Tasrkov and Horrocks, 2006) is an example of a automated reasoner relying on *SHOIQ* (printed in a *grotesque* typeface by convention) DL notation. This well developed reasoner is packaged with the widely known Protégé ontology editor (http://protege.stanford.edu). A popular KR language is the Resource

Description Framework (RDF) triple-based representation system, but it is considered by some to be semantically weak (Schulz and Jansen, 2013). Triple refers to the fact that relationships between concepts are represented by Subject-Predicate-Object (SPO) connectors where the object and subject are concepts, and the predicate is a relationship encoded in Description Logic (DL) syntax or an abstraction thereof. The specification of an ontology usually relies on a combination of the following Boolean constructors and restrictions (Staab and Studer, 2013; Handke, 2015):

Conjunction (∧, &, *and*)

Disjunction (∨, *or*)

Negation (¬, ~)

Equivalence (≡, ↔)

Material implication (→...⊃, *if*...*then*)

Universal restriction (∀)

Existential restriction (∃)

Applying ontologies

Once the ontological template has been populated with data instances, it becomes a Knowledge Base (KB). KBs serve as the foundation of the semantic web and they are a distinct form of databases in that the relations between data in KBs are semantically enriched. Although ontologies are different from database schemata for the same reason, ontologies can be useful in databases design. In essence, ontologies are re-usable and database schemata are once-off blueprints. It follows that ontologies have multiple purposes. Stevens (2000) lists these purposes as: providing a community reference system, defining database schemata, providing ontology based search tools, and supporting natural language processing. Examples of ontologically designed biological knowledge bases include: RiboWeb, EcoCyc, MBO, GO and TaO. BIOlogical PAthway eXchange (BIOPAX) structured language is an example of an effort to specifically control biological vocabulary to enhance communication between databases (Demir *et al.*, 2010).

Upper ontologies

Research is also being conducted into creating what are called *upper ontologies* (Soldatova and King, 2005). In contrast to subject specific ontologies such as biological ontologies, upper ontologies seek to anchor all other ontologies by serving as a root ontology. The Institute for Electronics and Electrical Engineering (IEEE) investigated the viability of such ontologies by creating the Suggested Upper Ontology Working Group (SUO-WG). They concluded their research by recommending further investigation into three candidate upper ontologies (Poli *et al*., 2010): the Information Flow Framework (IFF), the Suggested Upper Merged Ontology (SUMO) and the Upper Cyc Ontology (UCO). An example of a browser that was subsequently designed to navigate SUMO terms during ontology construction is SIGMA-KEE (http://sigmakee.sourceforge.net/). There is however a distinction to be made between ontology construction assistants such as SIGMA-KEE and the Protégé ontology editor mentioned earlier. Employing both these tools could assist the user in determining relevant standardized vocabulary from SUMO in the case of SIGMA-KEE, which can then be used to create structural and composition facets of an ontology in Protégé.

Ontology design obstacles

Typical issues that can arise during ontology construction are highlighted by example with Protégé ontology editor's accompanying Pizza tutorial (http://protegewiki.stanford.edu). These stumbling blocks include: determining valid membership of a class, avoiding ambiguity in assigning names to classes, defining classes using the correct quantifiers/properties, distinguishing a class from an instance, preventing duplication due to multiple inheritance, and separating structural from compositional relationships.

Biological database design

The data deluge

One of the most challenging aspects of biological database design is the necessity for continued maintenance due to information and software evolution. IBM (2012) estimated that 2.5 quintillion bytes of data was generated on a daily basis, and that 90% of the data in the world had been generated in the two year period leading up to the estimation. The bulk of new data is generated by the Internet of Things (IoT) as less human-to-human and human-to-machine interaction is required for information to be transferred over networks. This leads to an accumulation of stored information in every data warehouse imaginable including biological databases.

Structured, semi-structured and unstructured data

Blumberg and Atre (2003) estimated that structured data accounted for approximately 15% of data in existence. A more recent estimation by Cisco (2014) states that approximately 90% of data is either unstructured or semi-structured. Although there is disagreement about the percentage of structured data, there is a consensus that structured data is much less abundant than unstructured data. In computer science, data structures can be composed of primitive data types such as integers, composite types such as arrays, or abstract data types such as stacks. The type of a data element determines the set of values that the data element can represent (Wirth, 1985). Structured data can be readily read, but not necessarily semantically interpreted, by machines. It is *this* type of data that usually resides in databases, data warehouses, customer relationship management systems, enterprise resource planing systems, or extensible markup language (XML) documents (Cisco, 2014). In contrast, semi-structured and unstructured data is usually formatted as word processing documents (such as this thesis), emails, audiovisual files, or social media feeds. Semi-structured data can be created by organizing unstructured data into a hierarchy (taxonomy), or contextualizing unstructured data files with metadata (Blumberg and Atre, 2003), in a process called data classification. Automated data classification has grown in popularity over expert curated data classification due to the considerable time-savings that accompanies automation (Sebastiani, 2002). This transition has led to an increased interest in the field of machine learning (ML), which can be defined as automated general inductive processes, either supervised or unsupervised, that build classifiers. Software libraries such as Scikit-learn (Pedregosa *et al*., 2011) have been created to assist non-technical users with employing ML techniques in their research.

Broad classification of biological databases

Biological databases can be classified according to data type and use. Sub-categories that emerge from classification include sequence, structure, proteomic, interactomic and genomic databases (Cannataro *et al*., 2014). The information they contain represent a data layer between bioinformatics and molecular biology. The flow of information from the laboratory to data processing and interpretation can only be effective if the data model is appropriate. Traditional database design focuses on data models, declarative query language use, high throughput mechanisms, and reliability (Cui *et al*., 2014) More recently, designing databases with the aim of assisting users through data-centric decision making has been given a central role in development. This is likely due to the sheer volume of information that has become available. When databases grow in terms of size and count, four areas of concern become apparent: data storage and scope, processing capacity, level and method of digital curation, and data exchange facilitation (Zou *et al*., 2015). As mentioned in the introduction to this thesis, there are now more than 1500 human-related databases. So rather than attempt to provide a complete description of all the categories and sub-categories of biological databases, it would be more prudent to discuss projects that aims at indexing available databases. They can be conceptualized as databases of databases and will be referred to as *upper databases* for the rest of this review. The Nucleic Acid Research (NAR) upper database is called the Molecular Biology Database Collection (http://www.oxfordjournals.org/our_journals/nar/database/c/). This database had 1512 databases with fourteen main categories and 41 sub-categories indexed as at 2013 (Yu *et al*., 2015). On 26 August 2015 one additional main category could be viewed on the NAR upper database. The NAR publication has a Thomson Reuters impact factor rating of 9.112, a five-year impact factor rating of 8.867, and has been rated by the Special Libraries Association (SLA) as within the top 100 most influential journals. The 1512 databases indexed in the NAR upper database can therefore be assumed to be a conservative estimate. Some authors have even created indexes of upper databases. Bolser *et al*. (2015) tabulated ten upper databases as being similar in scope to their upper database called MetaBase.

Examples of small-scale experimental biological databases

Tangible User Interface (TUI) design has been put forward as a possible solution to help users cope with big data. This approach attempts to bridge the gap between the three-dimensional world and digital representations as a way to make user interaction more intuitive by *rich representations*. Eugenie++ is a TUI system that provides users with physical objects that can be manipulated on a horizontal multi-touch surface (Grote et al., 2015). The objects represent various navigational and logical methods that can be used to traverse the Massachusetts Institute of Technology (MIT) registry of biological parts and Pubmed. Genetalk is an example of a web application that serves as a communication platform by providing users with the opportunity to annotate positions on the genome with a diverse range of data in a conversation-thread like manner (Kamphans and Krawitz, 2012). In essence, each annotated variant becomes a blog and the scientific community then comments on each variant to exchange clinical and experimental findings with other individuals in the community. This could include users providing links to the literature, reporting annotation errors, or creating help request tickets. One drawback of Genetalk is that there is a heavy reliance on user participation for the curation and continued viability of the application. Moreover, no reference is made by Kamphans and Krawitz as to how the level of user participation was quantified. Determining user participation levels is important because previous studies (Ives and Olson, 1984) pointed out that there is not necessarily a positive correlation between user participation and information satisfaction.

Database schemata and data types

When a scientific problem has been formulated from observation and the manner in which data collection will be carried out has been established through experimental design, the next step for a scientist is to think about how data will be stored and retrieved for subsequent analysis. The database selection process should take into consideration the structure of the data that needs to be stored, but also the needs of the intended users of the data (Stonebraker, 2010a). The first step might be to define data types and their associated fields. This process is sometimes referred to as designing a database schema. The range of required data types will be influenced by the nature of the data to be collected and the manner in which the data will be accessed. Taking the project that this literature review motivates as an example, if a single alpha-numeric term is to be used during querying of a database, then a key-value hierarchical database will be more appropriated than a relational database. Similarly, if all data in a database can be declared as type *string*, a database management system (DBMS) could be prudently replaced with a flat-file database where type declaration and conversion can be easily manipulated by the designer using scripting languages. The main disadvantage of using a non-standardized database over a DBMS is that reliability is sacrificed for efficiency (Stonebraker, 2010a), that is, the following four components (Figure 2) of ACID-compliance is compromised to an extent:

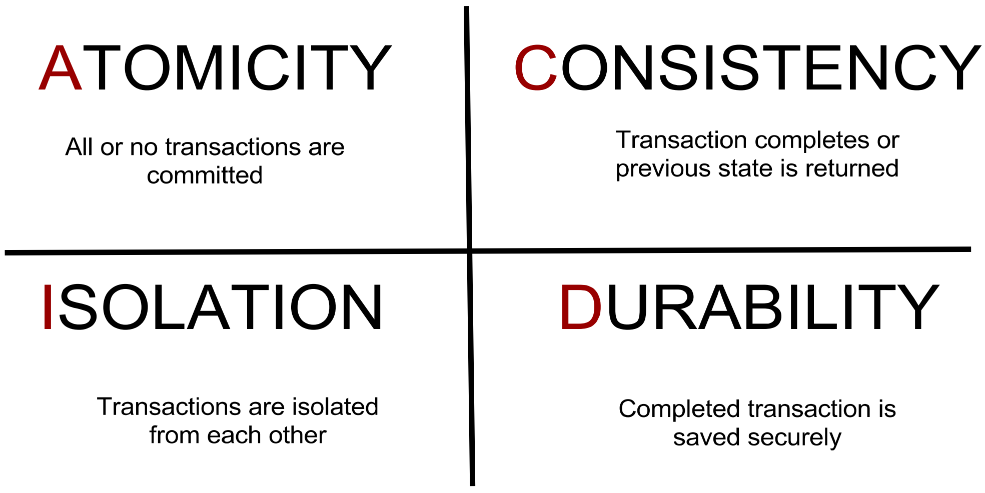

ATOMICITY
All or no transactions are committed
CONSISTENCY
Transaction completes or previous state is returned
ISOLATION
Transactions are isolated from each other
DURABILITY
Completed transaction is saved securely

Figure 2: ACID compliance ideogram

There is a significant overlap between ontology design and database schema design, specifically during construction of entity relationship diagrams (Kesh, 1995) and in a broader sense during construction of unified modeling language diagrams (Alkoshman, 2015). As mentioned earlier, ontologies are considered to be more generic than database schemata and it is for this reason that ontologies can supplement database design. Database schemata are intended to assist administrators and to some extent users in interpreting large databases not only during design, but also during maintenance, updating, and querying of such databases (Di Battista *et al*., 2002). Graphic representations of database schemata can decrease the complexity of lines upon lines of code. Di Battista *et al*. (2002) proposed an automated graphic representation framework that mediates XML data file submissions to a graph drawing application hosted on an off-site server. After the that has been processed on the external server, the same mediator then returns graphs of the submitted schema to the system hosting the schema. Graphviz

(Ganser and North, 2000) is an example of a comprehensive graph drawing suite that can be customized according to the needs of the developer. The Graphviz application is coincidently also offered as a plugin to the Protégé ontology editor that was discussed in the biological ontologies section. Database schemata fall into three major categories: relational, hierarchical, and graph based schemata (LinkedDataTools.com, 2015). Figure 3 illustrates the difference between the major database schemata.

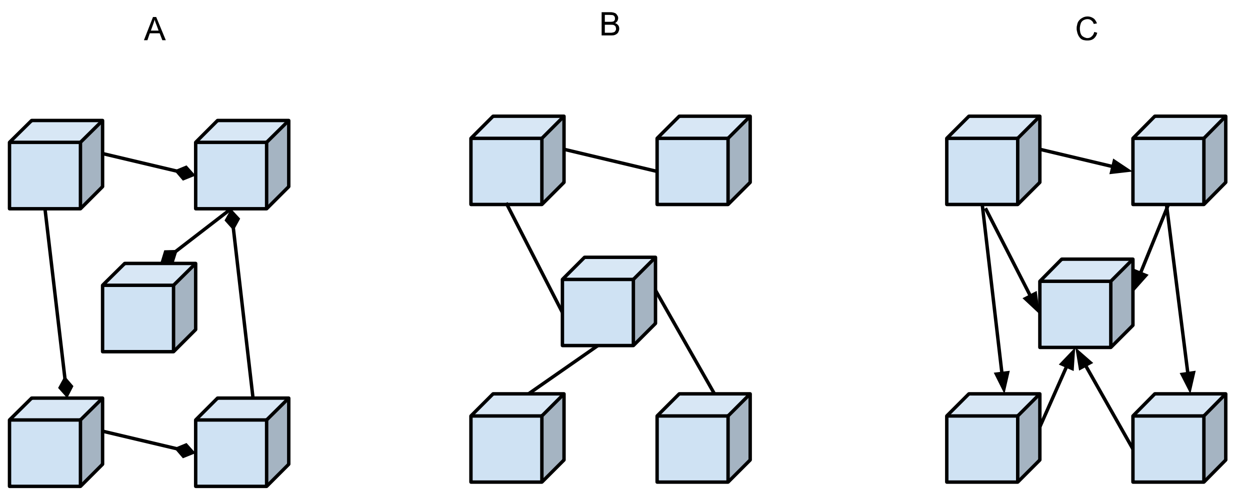


Figure 3: Graphical representation of the major database schemata: (A) relational (B) hierarchical (C) graph-based

The cubes in the figure represent classes, while the lines connecting cubes represent relationships between the classes. A diamond shape at the end of a line in Figure 1A indicates that the class to which it is nearest, is in a compositional ("has-a") relationship with the class connected to the other end of the line. The latter class is sometimes referred to as the parent class. The relationships depicted in Figure 3A are typical of those found in relational databases, where the classes represent the contents of tables, and the relationships indicate foreign key references. Figure 3B in contrast depicts a hierarchical structure where parent nodes have a more intrinsic value. Hierarchical database schemata characteristically adhere to the principal of orthogonality (Smith, 2008), which prohibits multiple inheritance, and seeks to minimize the amount of classes (Ross *et al.*, 2005). Graph based schemata, as illustrated in Figure 3C, are characterized by arbitrary relationships that do not make distinctions based on intrinsic importance. This type of schema is appropriate for a database that intend to place an emphasize on semantic web integration (LinkedDataTools.com, 2015).

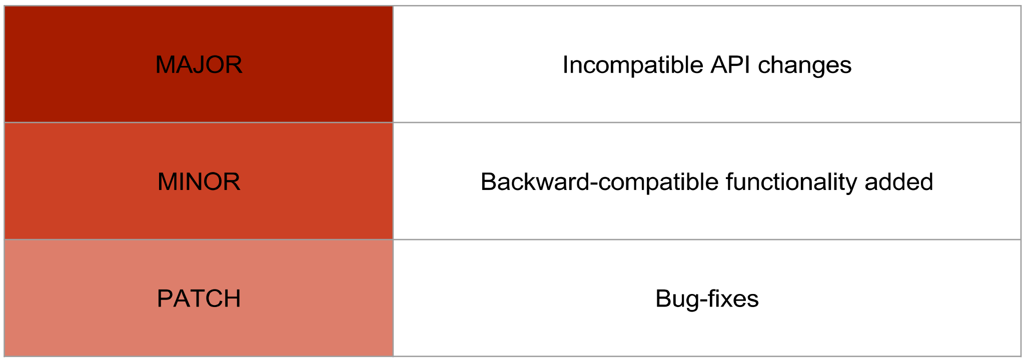
Relational databases vs NoSQL databases

Relational databases are best suited for storing structured data (Leavitt, 2010) and as such are queried using SQL. In contrast, non-relational databases (colloquially referred to as NoSQL databases) are designed to handle unstructured data more efficiently than relational databases. NoSQL databases can also be employed in distributed environments more easily than relational databases, thereby reducing the financial cost associated with isometric scaling. The three major categories of NoSQL databases according to Leavitt (2010) are: key-value, column-based, and document-orientated. Some authors make additional distinction by categorizing graph-based databases as a fourth type of NoSQL database (Tweed and James, 2010; Ponzanni, 2013). By comparing database schema categories as defined by LinkedDataTools.com (2015) in the previous section, with NoSQL categories as defined in this section, there seems to be a consensus that key-value and column-based NoSQL databases are most appropriately conceptualized with hierarchical schemata. Key-value databases are typified by nested key-value pairs, that is, each value acts as a column that can in turn be composed of a key-value pair. Conversely, column-based NoSQL databases typically have a single column that contains closely related key elements, while document-based databases do not put a constraint on the amount of columns.

NoSQL databases share the main disadvantage that non-standardized databases exhibit in that ACID-compliance is compromised. However, this shortcoming only becomes a concern when a database is distributed (Ponzanni, 2013). Brewer's theorem (more recently referred to as the CAP theorem) is an attempt to substitute ACID-compliance (Stonebraker, 2010b; Pokorny 2013) as a tool to judge the robustness of a NoSQL database.

Managing software dependencies

Large scale, distributed software applications require detailed planning of software deployment related activities that follow the end of a project's development phase (Dearle, 2007). These activities can be subdivided into an installation phase (release, configuration and activation of the software) and a post-installation phase (monitoring, updating, deactivation and redeployment). Smaller projects require less stringent planning of the post-development phase, but managing change remains important because responding to change is considered more important in the world of software development than sticking to a predefined plan (Manifesto for Agile Software Development, 2001). Software management can be complex and may include managing temporal and system properties, managing the properties of the object of change, and managing change events themselves (Buckley *et al*., 2005). Or, it can be limited by practical considerations to deciding on a versioning number scheme such as the Preston-Werner (Figure 4) semantic versioning scheme (Raemakers *et al*., 2014).



MAJOR	Incompatible API changes
MINOR	Backward-compatible functionality added
PATCH	Bug-fixes

Figure 4: Preston-Werner semantic versioning scheme

This scheme follows the MAJOR.MINOR.PATCH naming convention, with the first software release designated as v1.0.0 Each of the three digits will cycle up by one if the condition to the right of the respective categories in Figure 4 has been met. To ensure interoperability in a system that utilizes software from multiple contributors, dependency registers and automated package managers provide the ability to preserve the state of a set of software components (Abate *et al.*, 2011). Three exemplary package managers, that provide functionality in a Javascript environment, such as would be the environment for this project, are the general purpose Node Package Manager (NPM), and the front-end specializing package managers Bower and Component (Bevacqua, 2015).

User participation

The software development approach proposed for this project will be *mashup*-orientated rather than service-orientated or component-orientated. The main driver behind mashup development that distinguishes it from more conventional development approaches is that it places an emphasis on user innovation (Capiello *et al*., 2011). Mashup development is promoted by the availability of *open services,* such as APIs provided by, for example, Google and Twitter. In other words, during mashup development, the developer is also a user to some extent. This concept, of creating cycles of data re-use, is a cornerstone of what is popularly described as *Web 2.0* (O'Reilly, 2007). Capiello *et al.* (2011) further defines the characteristics of a mashup as: having a domain specific focus, providing an abstraction from detail, and providing immediate visual feedback on user actions.

User participation has long been considered essential to the success of software systems (Ives and Olsen, 1984), however, these authors also observed that proponents of user involvement rarely put forward empirically derived arguments based on strong theory. A three part series of papers discussing qualitative methodologies in the health care sciences, included a well argued paper by Reeves *et al*., (2008), that examines the use of ethnography in software design. In their paper, that acts as a guideline for determining behavioral, temporal and spatial dynamics in small populations, the authors suggest using nine observational dimensions during ethnographic research.

Semantic web integration

Communicating science

Biological scientists more often than not use prior knowledge to make inferences about the function of unknown entities rather than use axiomatic rules (such as contained in formulas and equations) (Stevens *et al*., 2000). That is why databases are important. In addition to the need for prior knowledge, scientists also need communication knowledge; they need to have reliable and efficient access to databases so that they can compare the data at their disposal. Ideally all of the available biological information should be integrated so that all of humanity's collected knowledge can contribute to future biological research. However, integrating databases is difficult because designing an all encompassing database inevitably leads to a series of compromises that sees a loss of information due to scientific and political ideals of different databases coming into conflict (Stein, 2003). A practical difficulty in establishing links between databases is encountered when attempting to put in place standardized naming conventions for biological objects and concepts. That is, the same biological object is sometimes given different names and different biological concepts are sometimes given the same name.

Data virtualization

An alternative to physical integration of databases is called data virtualization (also know as data federation). The central idea of this technology is that heterogeneous resources can be integrated by keeping track of the location of data (Machado *et al*., 2015), rather than housing the actual data. The inverse of data federation is query federation, where single or multiple queries are combined and sent to a predefined set of databases. In addition to this alternative to data integration, data interoperability can be promoted by technologies that allow the underlying data in databases to be connected. Knowledge discovery can be promoted by embedding HTML content with RDFa (Goble and Stevens, 2008) tags and with semantic metadata tags from standardized vocabularies such as Dublin Core (Weibel *et al*., 1998) and FOAF (Brickley and Miller, 2012). Data virtualization overlap with ontologies in a similar manner to how database schemata overlap with ontologies. The unifying similarity is the absence of actual data populations. Indeed, ontologies such as the Web Ontology Language (counter intuitively abbreviated as OWL) and OWL-S (Martin *et al*., 2004) have been specifically designed to encourage data integration using ontologies, and in the case of OWL-S where the “S” refers to “service”, to promote automated discovery, invocation and interoperability of such web services.

Linked data

Over the last thirty years, communication over the Internet, and before that over ARPANET, transitioned from flat files being exchanged with file transfer protocol (FTP), to the current day use of hyper text transfer protocol (HTTP) for implementing linked data (LD) communication structures (Sheridan and Tennison, 2010). LD is characterized by four defining principals as put forward by Tim Berners-Lee (who is credited as the inventor of the world wide web), namely: that uniform resource identifiers (URIs) should be used to identify real world objects, that these URIs are in HTTP notation to ease dereferencing, that the data should be enriched using RDF type standards, and that reference should be made to other objects within the data. The idea that modern data repositories should conform to the concept of LD has gained so much traction that a five-star rating system, devised by Berners-Lee (Janowicz *et al*., 2014), is frequently being used to gauge LD compliance. In this rating system, stars are awarded according to the following criteria:

⭐ The data is accompanied by human readable dereferencable data.

⭐ The data is accompanied by machine readable dereferencable data.

⭐ The applied vocabulary links to other vocabularies.

⭐ Metadata about the vocabulary is available.

⭐ Other vocabularies link to the applied vocabulary (contrast with third star).

Representational State Transfer (REST)

Since LD is inherently RESTful (Sheridan and Tennison, 2010), providing access to such data with servers that comply to REST constraints would, at least conceptually, decrease the complexity of such a service. These constraints of RESTful implementations was discussed in detail by Pautasso (2014) in terms of differing levels of maturity and with reference to the main concepts of addressability, stateless interactions, uniform interface, self-describing messages, and hypermedia. The latter concept supports the interpretation that LD is inherently RESTful, but extends it to differentiate web services that comply with RESTful constraints, by assigning to such web services the term hypermedia APIs. The web service that is being motivated by this review, will place an emphasize on making content discoverable, and could therefore be classified as an hypermedia API. A third group of researchers (Meng *et al.*, 2009), advocated the use of RESTful web services over traditional web servers, after empirically comparing their respective suitability in distributed data integration. RESTful web services seem to not have any particularly drawbacks according to the surveyed literature, however, REST technology relies heavily on the HTTP protocol which might or might not be replaced in the future.

SNPs: the determinant of genetic variation

Defining SNP based research

Genetics is the study of biological inheritance and subsequent variation among and within individuals, while genomics is more specific in that it concerns itself with the structure, function and mapping of genomes. A consortium led by Craig Venter revealed in 2001 that the human genome contains approximately three billion base pairs (Venter *et al*., 2001), with the bases being one of four naturally occurring nucleotides: adenine (A), cytosine (C), guanine (G) and thymine (T). There is a fifth abundant monomer of nucleic acid called uracil (U), but this monomer is a constituent of RNA, not DNA. Uracil, which is derived from the deamination of cytosine (Brown, 2007), is here referred to as a nucleic acid monomer entity because the nucleotide uridine is thermodynamically unstable (Peña, 2015). It is important to note that uracil is not a polymorphism of thymine in the context of the SNPs. Instead, substitution of A, C, G and T with each other in a strand of DNA represent SNPs. Together with restriction fragment length polymorphisms (RFLPs) and simple sequence length polymorphisms (SSLPs), SNPs are one of three types of DNA markers that are especially useful, and characteristically include at least two alleles (Brown, 2007). The quantity of SNP markers vastly outnumber that of other DNA markers because of its experimental through-put being higher than that of RFLPs (low though-put) and SSLPs (medium-throughput) (Scarano, 2014). SNPs arise as a results of DNA mutations, which in turn is caused by errors in DNA replication or as a result of mutagens acting on DNA (Brown, 2007). However, a second important note to make is that less than 1% of mutations that fall in the SNP category are functional (Venter *et al*., 2001), that is, they lead to a change in the function of a protein.

SNP related literature resources

Conventions used to represent elucidated genetic data differs from one resource to another. The International Cancer Genome Consortium (ICGC) (https://icgc.org/icgc), lists data categories for variation together with their respective entry availabilities, and contains sample generated information on 12979 donors. Of these, the database has simple somatic mutation (SSM) category data for 8038 individuals, making the ICGC seems like the ideal place to search for literature on a given SNP using its identifier (e.g. rs1799950 –

an SNP in the *BRCA1* breast cancer related gene). However, a search using this refSNP returns no results from the ICGC. When searching for this term using a more generic database such as dbSNP (that forms part of NCBI and therefore the Pubmed and Pubmed Central resource repositories), relevant results are returned for a plethora of information after which data in related NCBI databases can then be retrieved by selecting a database from a drop-down list. Selecting Pubmed from the drop-down list provides the user with a subsequent list of articles that relate to the SNP, with the option to navigate to related articles, however, no indication is given as to the relevance of the article to the identifier if more than one article is associated with the identifier. Instead, literature is sorted with a preference for more recently publicized articles (as discussed earlier in this review). Searching for the SNP by its identifier in non-institutionalized databases such as SNPedia (http://www.snpedia.com) returns results that point the user to literature relevant to the SNP, with an indication of whether this literature is provided under Open Access agreements, but the order in which the results are listed does not give the user an indication of how relevance was determined. In addition, the literature that SNPedia refers the reader to does not give an indication of the date of publication. Interestingly, SNPedia does link the identifier with keywords and provides quantification by mentioning other SNPs in context (e.g. "This SNP, a variant in the BRCA1 gene, is 1 of 25 SNPs reported to represent independently minor, but cumulatively significant, increased risk for breast cancer."). Specific literature ordered by quantified relevance is not supplied by SNPedia, and bulk querying using reference SNP identifiers is not available as part of its user interface.

VCF files

According to Venter *et al*. (2001), the human genome contains approximately four million polymorphisms (DNA bases with at least two alleles). This is much lower than the three billion base pairs present in the human genome, and a specialized file format containing only data on the occurrence of variation would therefore remove 99.8% of data generated during sequencing and found in, for example, FASTQ files (Cock *et al*., 2010). Towards this end, variant call format (VCF) files were developed to cater for variant analysis, not only among individuals, but also across multiple samples (Danecek *et al*., 2011). This file format was specifically developed for the 1000 Genomes Project, where the need for removing redundant raw sequencing data became necessary to efficiently compare results from the sequenced genomes of 1000 individuals. The following chart (Figure 5) illustrates the importance of reducing file size by plotting 1000 genomes compressed VCF file sizes as a function of human chromosome numbers:

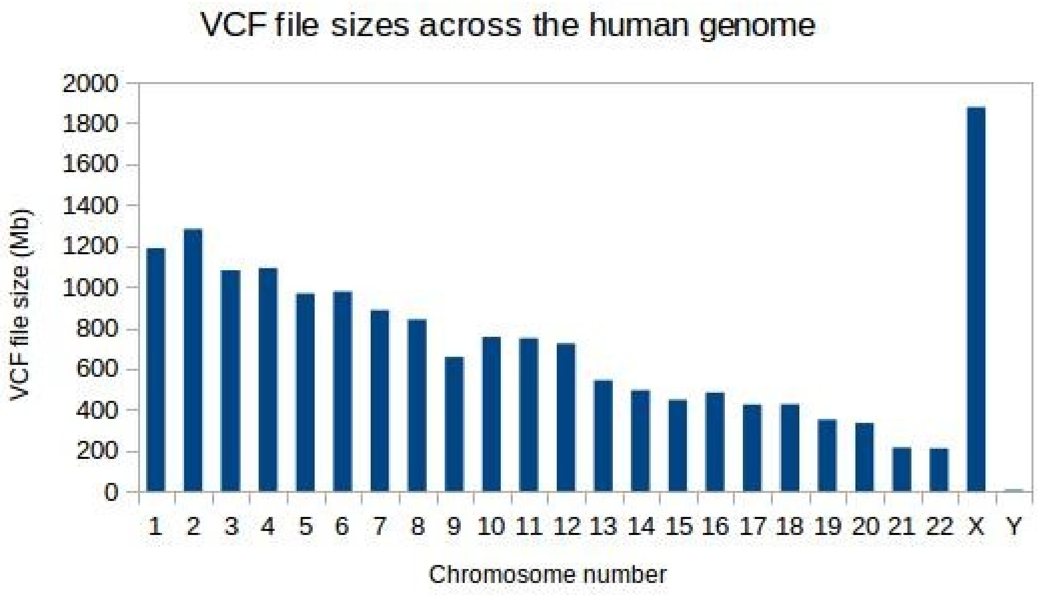


Figure 5: VCF file sizes for each of the 46 (23 x 2) human chromosomes

The above chart was compiled using data from the 1000 Genomes Project repository (ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/). The resulting plot depicts an overall decrease in data volumes across a sequential range of the human karyotype. This decrease correlates with a decrease in chromosomal length (not depicted). The overall size of VCF files indicate the need for removal of non-variant data, not only for the 1000 Genomes Project, but also for any other projects exceeding 1000 individuals. Although VCF files are more succinct than raw sequencing data in the context of variation, further parsing could be of benefit when, for example, only the ID column of a VCF file is of interest to the researcher. Software that allows for further parsing could be in the form of a standalone package, such as PyVCF (http://pyvcf.readthedocs.org/en/latest/index.html), or integrated with online bioinformatics platforms such as Galaxy (https://galaxyproject.org/).

Open access initiatives

Defining and motivating free articles

Open access (OA) was defined by the Budapest open access initiative (BOAI) convention of as follows:

The literature that should be freely accessible online is that which scholars give to the world without expectation of payment...
(BOAI, 2002)

Publishing models based on this believe is becoming increasingly popular. Official figures show that OA articles are increasing at a rate of about 10% per annum and an estimated 71% of biomedical research articles published between 2011 and 2013 are currently available through OAIs (Archambault *et al*., 2014). In their report on OA proportions to the European Commission, these authors ascribed growth in the amount of OA papers to four drivers: increased interest in OA leads to more new papers being published under OA licenses, more paid papers becoming OA licensed for the same reason, expiration of embargo periods during which access to scientific literature is restricted, and an increase in the amount of overall published scientific papers per annum. The increase in freely available articles represents a new source of information for scientists. This is because paid-access models often only give free access to the abstract of an article, while in open access initiatives the reader has free access to the entire article. Shah *et* al. (2003) concluded that when creating a subset of literature for a database, access to the full text of an article is preferable to only having free access to the abstract. The following illustration (Figure 6) highlights the crux of their finding by incrementally saturating the dispersion of gene names per article subsection based on the likelihood that a gene name will appear in a given subsection:

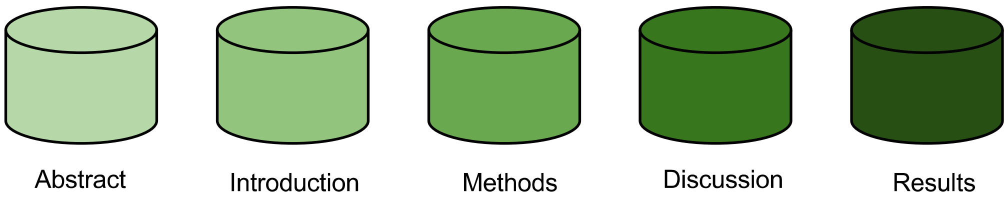


Figure 6: Gene name occurrence frequency depicted with incremental saturation (after Shah *et al*., 2003)

If cost is a limiting factor for a reader who is interested in determining the relevance of thousands of articles based on the occurrence of a gene of interest, as was the case in the study that the data for the above figure was drawn from, availability of free access to to all subsections of an article will be a primary determinant of the feasibility of a project.

Open access categories and licensing

The quality of free publications can vary widely and therefore OA material is often placed into the following categories: gold (e.g. listed in the Directory of Open Access Journals), green (listed in directories such as OpenDOAR that caters for self-archived material), others (papers available through large databases such as CiteSeerX and Pubmed Central) and rogue material (papers published without consent (see *US vs. Aaron Swartz* (2013))). The legality of OA is ensured by obtaining a copyright owner's consent with a license, such as the Creative Commons license, that may allow users to read, copy, download, link, crawl, and search articles in OA repositories (Suber, 2004). Persuading researchers to provide their written works under such licenses is not considered a hurdle to overcome since there is a long standing tradition in the scientific community to write articles for impact rather than profit. This tradition dates back to the creation of the first scientific journals in 1665 and benefits science by increasing the scope of publication through reduced publication cost. However, the current percentage of green open access journals only amount to between 10% and 20% of all published articles (Harnad *et al.*, 2008), with green OA papers constituting 90% of all OA material. Employers and funders who commit to mandating self-archiving could have a significant impact on the availability of green OA in future. OAIs occasionally rely on open source software and software technicians donating small amounts of their time. Suber (2004) mentioned that OAIs can be financially maintained by being granted space on university servers. Hosting of OA material in this way is not considered to be completely altruistic since the hosting institution gains by increasing its research output and visibility. The Pubmed Central open access initiative (PMC-OAI) is an example of an OAI repository using the open archives initiative protocol for metadata harvesting (OAI-PMH). It contains partial (or complete) access to articles published in a list of journals available at: https://www.ncbi.nlm.nih.gov/pmc/journals/

Proponents of open access

OA literature has been shown to have a greater research impact than literature based on reader-pays models (Antelman, 2004). According to the Australian open access support group (AOASG) (2015), there are eight properties of OA that makes it a viable publishing model: OA papers get higher citation rates, which leads to researchers gaining increased exposure (Harnad *et al.*, 2008; Mazloumian *et al*., 2011), access is made available to readers in developing countries (Chan *et al*., 2005), taxpayers get better value for their money (Harnad *et al.*, 2008, Phelps *et al*., 2012), research input to practical application output is better served (Willinsky, 2005), the public becomes informed about scientific activities (Arzberger *et al.*, 2004, Willinsky, 2005), OA research is compliant with grant rules (Harnad *et al.*, 2008), and OA can influence public policy (Willinsky, 2005; Mazloumian *et al*., 2011).

Critics of open access

Concern is often raised about OA by suggesting that subscription services will be deprecated by free publications (Aronson, 2005), that OA will encourage or allow bypassing peer-review (Hunter, 2005; Haug, 2013), that it deprives authors of royalties (Goodman, 2004; Hunter, 2005), that it invites plagiarism (Goodman, 2004), and that it will become a haven for poor quality and rejected material (Goodman, 2004; Hunter, 2005;Haug, 2013), where conflicts of interested and copyright infractions are commonplace (Salem and Boumil, 2013). The argument that OA material in general is of inferior quality could be due to an unwillingness to recognize the distinction between OA categories and how peer-review processes are handled in each of these categories (Suber, 2009). In other words: the gold, green, other and rogue categories of open access material often seem to be erroneously equated with each other.

Ethical considerations relevant to biological data

Data privacy, ownership and consent

Even though abuse of personal records created during medical research on health and disease has not been documented (Gulcher *et al*., 2000), and is speculated to occur at most very rarely, it is still necessary to consider the impact of information management on privacy due to the legal implications that accompany violation of a person's right to privacy. This is especially relevant in the field of human genetics where governments, insurers or employers could theoretically discriminate against an individual based on that individual's genetic predisposition (Rindfleisch, 1997). To reduce the possibility of research data being used unscrupulously, various oversight committees have enacted best practices guidelines. The Icelandic data protection commission has, for example, outsourced encryption of all data gathered during disease based gene discovery to a third party (Gulcher *et al*., 2000). Their system ensures that data generated in the laboratory is de-identified before publication. Research has also been conducted into minimizing information loss during the process of de-identification, which has led to proposed solutions to the k-anonymity and l-diversity computational problems encountered during the process of automated de-identification (Ghinita *et al*., 2007). In addition, watermarking algorithms traditionally used for copyright protection of media files have been integrated with binning algorithms to extend copyright protection to medical records (Bertino *et al*., 2005). Increasing demand for secondary use of medical data necessitates the need for protecting data owners while simultaneously still making it possible to determine the provenance of medical data. This complexity is compounded by the infeasible of obtaining specific consent for secondary use of medical data as highlighted by O'Neill (2003) in his examination on the limitations of informed consent.

Access to information and equitable treatment

...people have a critical stake in how experimental results affect their health, personal economy, and quality of life...
(McInerney *et al*., 2004).

In the United States, patients have a right to review their medical records and this has led to the creation of online portals such as the Patient Clinical Information System (PatCIS) that makes it possible for patients to view their medical information over the internet (Cimino *et al*., 2002). The creators of the PatCIS system reported that no adverse effects were observed by extending the right of patients to view their medical information from hard copy access to access through the world wide web. Giving people the opportunity to access their medical records online could solve an observed inequality (Aday and Andersen, 1974) in access to healthcare between urban and rural individuals. Moreover, it would be easier to quantify the behavior of individuals who review their records online and that in turn would show support for the access-quantification-concept put forward by Donabedian (1972) that states that the use of a service rather than the mere existence of a service is proof of access.

Conclusion to the literature review

The motivation for this study arose from a general observation, noted as far back as the 1980's by some authors, that management of biological data rather than generation thereof has become the central area of concern for biologists and indeed for researchers and practitioners in other fields. Within the regional context that this project will be carried out, a suggestion was made that the expected increase in raw sequencing data from South African scientific activities, and specifically from the South African Human Genome Programme, should be supported by investigation into how subsequent data will be made available to the scientific community. The literature confirmed that SNPs are by far the most abundant DNA marker and that SNP data is often represented within VCF files, the latter being standardized by the well known 1000 Genomes Project as the medium of choice for representing biological variation extracted from raw sequencing data. RefSNP identifiers in the ID column of VCF files were recognized as the key terms of reference for SNP based variation, and as the probable query term when a scientist might conduct further research based on SNPs.

Having determined a specific focus for the project on SNP data within VCF files, further analysis of the literature revealed that there is a need to increase research efficiency when using unstructured data. An opportunity was recognized in the increased availability of open access literature that could serve as a source of semi-structured data. This data could in turn be integrated with search engine query constructs. An argument was made that for an integrated service to be effectively deployed, it has to be aimed at complying with integrative technologies such as *Web 2.0,* semantic web technology, and linked data principals. The overarching influence that ontology has on each of these technologies was demonstrated by contrasting ontologies with database schemata and pointing out similarities with linked data principals. Moreover there was an overwhelming consensus in the literature that RESTful based web services architecture should be applied when designing *mashup* artifacts with an emphasize on semantic data integration.

Examination of the current state of the literature has confirmed that there exists a gap in post variant call research that could feasibly be addressed by implementing a software system that integrates structured data with open access literature using free and open source software.

METHODS AND MATERIALS

User surveys

Prior to the start of the project's development phase, a survey was conducted to determine the needs and opinions of biological database users at the South African National Bioinformatics Institute (SANBI), the Institute for Microbial Biotechnology and Metagenomics (IMBM), and the Molecular Virology laboratory at the University of the Western Cape (UWC). The survey was drafted using a Google Forms template obtained from https://www.google.com/forms. An open invitation to complete the survey was sent by email to the three facilities just mentioned as well as to a few individuals not associated with these facilities. The questions in the survey were designed to determine temporal, spatial and behavioral parameters in nine observational parameters as per the guidelines set out by Reeves *et al*. (2008), but adapted for surveying biological database user experience. A post-implementation survey was carried out in a similar manner to determine whether the developed application met user expectations. Graphs of the resulting data were drawn using Libre Office Calc v4.2

Database construction

Article sourcing

Scientific articles made available for public use by the Pubmed Central open access initiative (PMC-OAI), were downloaded in extensible markup language (XML) format from PMC's database. 1,132,084 articles were downloaded in four zipped batches from http://www.ncbi.nlm.nih.gov/pmc/tools/ftp/ Retrieval was carried out from an Ubuntu v14.04 terminal using GNU Wget v1.15. The URL just mentioned, was then concatenated with the file name of each of the respective four batches, and passed as parameters to the Wget command.

Extraction of articles containing SNP identifiers

A Python v3.4.3 script (Addendum, script 1) was written to scan through the downloaded articles in search of articles containing at least one occurrence of a reference SNP identifier. For the rest of this thesis, Python will refer to Python packaged with the Anaconda v2.2.0 distribution. Reference SNP identifiers were taken to be r*eference SNP cluster IDs* (accession numbers used by databases such as dbSNP). These identifiers were recognized by searching through the articles word for word, and if a word started with “rs”, or the second and third characters were “rs”, the file in which the word occurred was then copied to a separate directory. The second disjunctive clause was necessary since authors occasionally write refSNPs in brackets. To decrease algorithmic complexity, control was passed to the next iteration upon the discovery of the first valid term in a file. Counters were also instantiated to keep track of the total amount of files scanned and copied.

Parsing of articles containing reference SNPs

The next script (Addendum, script 2) was written to extract information, needed for the search engine, from the subset of the literature corpus that contained refSNPs. Unicode category C characters (http://www.unicode.org/reports/tr44/#GC_Values_Table) were excluded with the `remove_control_characters` helper function. This function also removed commas and backslashes to prevent parsing errors resulting from ambiguous interpretation of commas (in Javascript object notation (JSON)) and backslashes (in the Python language). The following information were extracted from each article containing at least one reference SNP cluster ID: the refSNP, the article title, the publication date, the email address of the correspondence author, the author defined keywords, the article's digital object identifier (DOI), and the article's Pubmed Central ID. The extraction was repeated for every occurrence of a refSNP within a given article so that a refSNP together with it's associated data could be converted as a completely encapsulated object to JSON. Since there is currently no upper limit on the length of a refSNP, in terms of the amount of characters a refSNP may contain (NCBI Bookshelf, 2014), the distribution and upper limit of refSNP character lengths were quantitatively determined for the current corpus to elucidate trends in refSNP naming conventions. The character length of refSNPs for inclusion in the database were subsequently capped at six characters at the lower end and twelve characters at the upper end. This most appropriate interval was determined by manually checking the validity (unambiguity) of low character count refSNPs and determining the maximum character count of a refSNP using preliminary data from the extraction process. After extraction of all relevant information, each refSNP was concatenated with its associated terms using tab delimitation (after stripping surrounding white space from all the terms), and then the concatenated string was appended to a master list as a single object. Next, a dictionary was created to accept entries from the master list as keys, and an incremental occurrence counter updated the value of each dictionary item, which ended up serving as an indication of how many times a refSNP has occurred in an article. Each dictionary key-value pair was then concatenated with tab delimitation, and written to a comma-separated values (CSV) file, in preparation for conversion to JSON format.

Javascript object notation encoding

The third python script (Addendum, script 3) was used to convert the tab delimited data generated in the previous step to JSON. A loop in the script added data from each line in the CSV file to a string after concatenation of the appropriate JSON punctuation marks. The first level key was added prior to iteration. After iteration was completed, the end of the string was sliced to remove an artifact created during the final iteration, and closing JSON was then added to the string. Finally, the string was written as a single line to a file with the name `JSONdbKeys.json` This file served as the database for the project.

Creating an index of author defined keywords and reference SNP cluster IDs

A front-end predictive text feature, that will be described in more detail later, required the generation of two sets of arrays containing a complete set of author defined keywords and extracted refSNPs. These arrays were created with the use of a Python scripts (Addendum, script 4 and 5) during the process of preparing data for inclusion in the database. Both these scripts extracted data from the file created during the previous encoding step. The algorithms started by determining the amount of unique records within the database. This was done to avoid out of bounds errors. Alphanumeric single terms or phrases that didn't start with “rs” were extracted to create the keyword array. Both algorithms created sets to exclude duplicates, and then sorted the sets as a Python specific mechanism to convert the sets back to lists (arrays). The data was the copied to Javascript files that were in turn imported into the relevant HTML pages.

Server side development

Instantiating and exposing a server to the web

The database file was served with the use of JSON-server v0.7.20 installed on a virtual machine provisioned on the South African National Bioinformatics Institute's mainframe. Running JSON-server required the installation of Node v0.10.25 as an interpreter. During development phase, the server was started from the command line by passing the initialization script and database file name as arguments to a node command. However, once the project was moved to production phase, two additional packages were used to run the application as a service on Ubuntu v14.04 as follow: First, the Forever v0.15.01 node package was installed that allows scripts to run as a daemons. Afterwards, an add-on to this package called Forever-service v0.5.4 was installed to start the database server as a system registered service. Both these packages were installed using the Node Package Manager (NPM) v1.3.10 with an optional parameter to make the packages available globally instead of installing the packages locally. To register the service with the operating system the following command was passed:

```
sudo forever-service install sniphunter --script index2.js
```

The application was then started with:

```
sudo start sniphunter
```

Running the above command started JSON-server at port 3000 (according to JSON-server internal specifications) and it was therefore necessary to enabled port forwarding using the following two commands to allow interaction with the service when a client connected at port 80 (the standard HTTP port):

```
sudo sysctl -w net.ipv4.ip_forward=1

sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-port 3000
```

By enabling port forwarding (first command) and then creating a port forwarding chain rule using iptables (second command), the JSON-server API and the website developed during this project was exposed as a web service and web application, respectively.

Implementation of a customized JSON-server wrapper

Custom changes to the JSON-server were made using a wrapper (Addendum, script 6) supplied with the JSON-server source code. The following three packages were installed using NPM for use in the wrapper script:

- Awk [a wrapper of WebAwk] v1.0.0
- Intercept-stdout v0.1.2
- Multer v1.1.0

In addition, the Filesystem package bundled with the Node interpreted was also imported for use in the wrapper script. All packages, with the necessary exception of Multer, were limited in scope to their local environments to avoid possible conflicts with the JSON-server package. Three custom routes were implemented with the use of these four packages. A POST route was set up using the Multer package to allow users to upload VCF files to the server for parsing. The uploaded file was then saved to a temporary folder and the path to this file was passed to a global variable. Two GET routes were then implemented to handle file parsing. The first GET route used the Filesystem package to create a file object to pass as a parameter to a method of the Awk package, with the following command being passed as a file object to the same method:

```
$3 ~ "^rs" {print $3}
```

The intercept-stdout package was used to redirect Awk output. The redirected results were then written to a file and the file was returned to browser. The second GET route was similar to the first except that headers for information relevant to genome co-ordinates were included in the output by amending the AWK command as follows:

```
BEGIN
{printf format = "%-15s %-15s %-15s\n",
"Chromosome","Position","refSNP_ID"; printf format = "%-15s %-15s %-
15s\n", "---------","-------","--------" }
$3 ~ "^rs" {printf format, $1 ,$2 ,$3}
```

Client side development

HTML structure

HTML was used to structure the content of the web application being server by the JSON-server. Seven main HTML pages were created to serve the various features offered by the web application: a single refSNP search page, a multiple refSNP search page, a keyword search page, an API usage guideline page, a VCF file parser page, a software download page, and a contact page. In addition, an upload confirmation page, a page containing all keywords in the database, and a page containing all refSNPs in the database were added to the public directory. The keyword and refSNP pages were added to allow automated web crawlers to discover and create links to the service using content within the database. All pages, with some exceptions in the confirmation page, were structured according to the following pseudocode:

```
#Declare the document type
#Create opening HTML tag and opening head tag
#Add metadata and title tags
#Add relevant external script and stylesheet references
#Add in-page Google analytics tracking script
#Create closing head tag and opening body tag
#Add RDFa metadata
#Add navigation bar and modal content (if applicable)
#Add main content
#Create closing body tag and closing HTML tag
```

All pages contained a navigation bar (navbar) with the ability to collapse the navbar added by using a built-in Bootstrap class. This made it possible to hide page items in the navigation bar when the user had a smaller screen size. Once the page items were hidden, they could be viewed by clicking on a trigram in the top right hand corner of the screen, which would open a drop down menu containing the page items. The first page item on every navbar contained the SNiPhunter brand name in a distinct font.

All main pages of the website contained open access and open source logo's above the main heading. Links to external resources with more information on each of these topics were embedded within the images. Statistics on the amount of database entries were included below the main heading on pages that returned dynamic content from the database. Queries on these pages could be launched by clicking on the paper plane glyphicon or by pressing the enter key. A refresh button was added to clear old results and refresh forms once clicked. In addition to the predictive text future described in more detail in the layout section that follows, hints were also added below some form element input boxes to give the user additional guidance in using the web application. Moreover, information buttons were included on pages that required user input. When clicked they would open a modal with usage information, and links to template text (TXT) and VCF files, where applicable. A green information glyphicon was overlaid on the info button to make it stand out for new users.

Web application layout

Bootstrap v3.3.5, together with a single custom cascading style sheet (CSS) (Addendum, script 7), was used to style the web application. The Bootstrap library was downloaded and stored in a local directory instead of using an external content distribution network (CDN). Both the Bootstrap library and the custom style sheet was imported into all HTML documents using the following tags in the head section of the respective HTML pages:

```
<script language="javascript" type="text/javascript" src="/scripts/jquery-
2.1.4.min.js"></script>
<script language="javascript" type="text/javascript" src="/stylesheet/bootstrap-
3.3.5-dist/js/bootstrap.js"></script>
<link rel="stylesheet" href="/stylesheet/bootstrap-3.3.5-
dist/css/bootstrap.css">
<link rel="stylesheet" href="/stylesheet/mystyle.css">
```

Since the Bootstrap library has jQuery as a dependency, jQuery v2.1.4 was downloaded, stored in a local folder, and included in HTML pages prior to the importation of the Bootstrap library. The content of the custom style sheet included two fonts (Architect's Daughter and Comfortaa), that were applied to the content of the HTML pages by using a custom class, or by overriding fonts used for paragraphs and headings. The loading of a wallpaper displaying three nucleotides (sourced from the Internet and adjusted with GNU image manipulation program (GIMP) v2.8.16), was also accomplished using the custom style sheet. An Ajax loader (https://commons.wikimedia.org/wiki/File:Ajax-loader.gif) and image rotater (http://codepen.io/fivera/pen/uLgyj) were also included within the stylesheet.

A CSS library, that formed part of the Awesomplete v0.0.0 predictive text package, was downloaded from Github and imported into the search feature pages:

```
<script language="javascript" type="text/javascript" src="/scripts/awesomplete-
gh-pages/awesomplete.js"></script>
<link rel="stylesheet" href="/scripts/awesomplete-gh-pages/awesomplete.css" />
```

Custom styling of this library, to complement the theme of the website, was carried out by directly tweaking parameters within the Awesomplete CSS file.

Javascript interactivity

Native Javascript functions and jQuery v2.1.4 were used to allow user interaction with the web application. In contrast to the VCF file parser page that uploaded a VCF file to the server, a text file uploaded via the multiple refSNP search was not sent to the server. Instead it was handled by the browser on the client side. The files were handled differently because manipulating a VCF file and generating a new file from the resultant data required the use of file system processes than are not available on the client side. The VCF file parsing page had an action assigned to an express route inside a form element. This action was triggered once a file was selected and the upload button clicked. The multiple refSNP page had four functions (Addendum, script 8) for text file handling: a function for detecting a file selection event, a function for converting the file to UTF-8, a function for presenting the data to the user for verification, and a function to handle a file read error. With both the refSNP search pages and the keyword search page, queries were sent to the database using the following jQuery AJAX call:

```
$.ajax({
url: root + '/PMCOAI_rs_articles?rs_number=' + input,
method: 'GET'
}).then(function(data) {});
```

The root variable in the AJAX call was bound to localhost in the development environment and http://sniphunter.sanbi.ac.za in the production environment, while the refSNPs were bound to the input variable. In the case were multiple queries were launched, an iterative function looped over the rs numbers and assigned them individually to the input variable with each successive AJAX call. The return data object was then used to create text nodes objects ,and the text node objects were then sorted according to the amount of times a refSNP occurred in an article. Next, the HTML document was dynamically updated using the following two functions:

```
//DOM update if no results found
if (listz.length == 0) {
var elem = document.createElement("h4");
var noresults = document.createTextNode("\"" + input + "\""
+ " did not match anything in the database");
elem.appendChild(noresults);
$("rs").append(elem);};

//DOM update if results found
while (listz.length != 0) {
readyToGo = listz.pop();
readyToGo = readyToGo.concat();
$("rs").append(readyToGo[1]);
$("[id=updated]").addClass("btn btn-default updateSpecs");};
```

The remaining code in the query scripts handled the refreshing of upload forms and clearing of results if the user wished to restart the query process.

Linked data and search engine optimization

Dublin core (http://dublincore.org) vocabulary, HTML metatags (http://www.metatags.org) and resource description framework attributes (RDFa) (https://schema.org) were used to create machine readable content for inclusion in the source code of web pages. Standard HTML metatags were used to set the character encoding for the web pages to UTF-8, to provide a description of the page that could serve as a blurb in search engine results, and to tag each page with relevant keywords. Dublin Core (DC) vocabulary was used to provide the title of the web application, to inform crawlers that the subject is bioinformatics, to tag the data as created by Werner Veldsman and as published by SANBI, and to provide a date of publication. RDFa were added to inform search algorithms that the web application was of a medical nature, and that the target audiences were medical researchers and clinicians.

To further promote the discoverable of SNiPhunter, a sitemap was created using *XML sitemaps* (https://www.xml-sitemaps.com/), leaving all default settings unchanged. The automatically generated sitemap was added to the root directory of the public folder. The sitemap was named *sitemap.xml*, which is a convention used to indicate to automated web crawlers that this file contained information on the website's structure. In addition, the sitemap was uploaded to Google Webmaster Tools to make the search engine explicitly aware of the SNiPhunter web application and its internal links.

Website and user analyses

The following tracking code was included in each page, except for the confirmation page, to gather usage information when users interacted with the web application:

```
<script>
 (function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
 (i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
 m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
 })(window,document,'script','//www.google-analytics.com/analytics.js','ga');
 ga('create', 'UA-68718403-1', 'auto');
 ga('send', 'pageview');
</script>
```

Information obtained with this script included the country and city from which the user accessed the web application as well as details on the operating system, browser, Internet service provider, and screen resolution of the device used to access the site. Moreover, the behavior of the users could be monitored by gathering information on the landing pages, search engine queries made, and whether the users accessed the site directly, by organic search, or by referral from another site. All the preceding information was analyzed in terms of acquisition (total sessions vs new user sessions) and behavior (bounce rate, pages viewed per session, and average session duration).

Software testing

Query response time

Query response time was measured ten times for an refSNP that returned a single result and ten times for a refSNP that returned two results. Query response time was also measured ten times during the making of a single, double and triple query. All measurements were made using the network feature of Firefox Web Developer tools. Graphs of the data were drawn with Python's matplotlib v1.4.3 library. Regression analysis was carried out using the numpy v1.9.2 library for the Python interpreter.

Use case test

A random subset of refSNPs was selected to confirm the validity of returned results by constraining the selection to: a refSNP with minimum length, a refSNP with maximum length, a refSNP with average length, and an invalid refSNP. The following validation criteria was used during this test:

- Did the refSNP appear in the predictive text feature?
- Did results point to the correct internal resources?
- Did results point to the correct external resources?
- Did HTTP GET calls to the database return data objects to the browser?
- Could API calls be made to the database using Python?
- Were no results returned if the refSNP was not in the database?

Queries based on keywords used the same validation criteria, but different constraints. A single keyword, a phrase, a non-valid keyword, and an empty string were tested.

Web application performance measurement

Google PageSpeed Insights was used to monitor the performance of the website on mobile and desktop devices. This tool monitored the time that elapsed since a user requested a page to the time that the first view was rendered (above-the-fold content) as well as to the time the page was completely rendered. The results returned from this network-independent assessment was then reviewed to determine whether any changes with regards to server configuration, HTML structure, and use of external resources were necessary. WebPageTest (http://www.webpagetest.org) was used to supplement results from Google PageSpeed Insights with first view, repeat view and content breakdown charts.

SNiPhunter installation

SNiPhunter's source files were downloaded from Github using the following command from an Ubuntu terminal with Git v1.9.1 installed on the operating system:

```
git clone https://github.com/Werner0/SNiPhunter.git
```

Installation and setup was completed using the guidelines provided on the Github repository, and the use case test was repeated.

RESULTS

Pre-implementation survey

Eighteen biological database users completed the pre-implementation questionnaire. The following graphs (Figures 7 and 8) depict results of two of the ten questions posed to them during the survey:

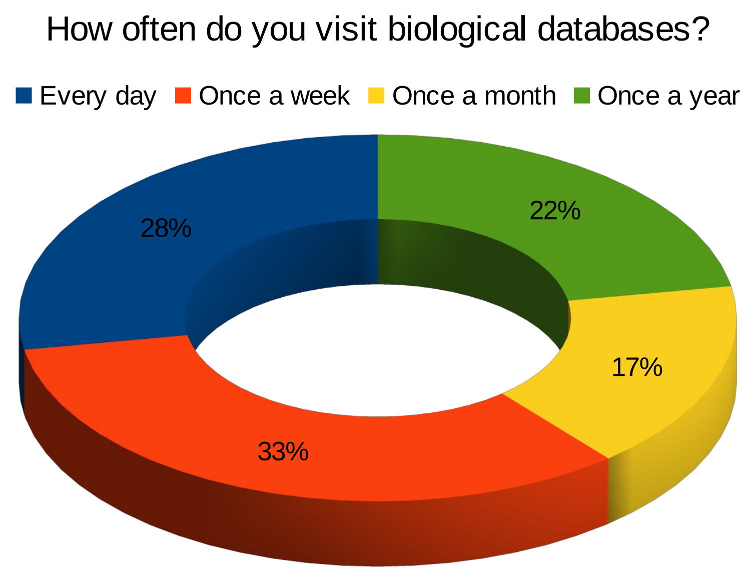


Figure 7: Temporal biological database user behavior

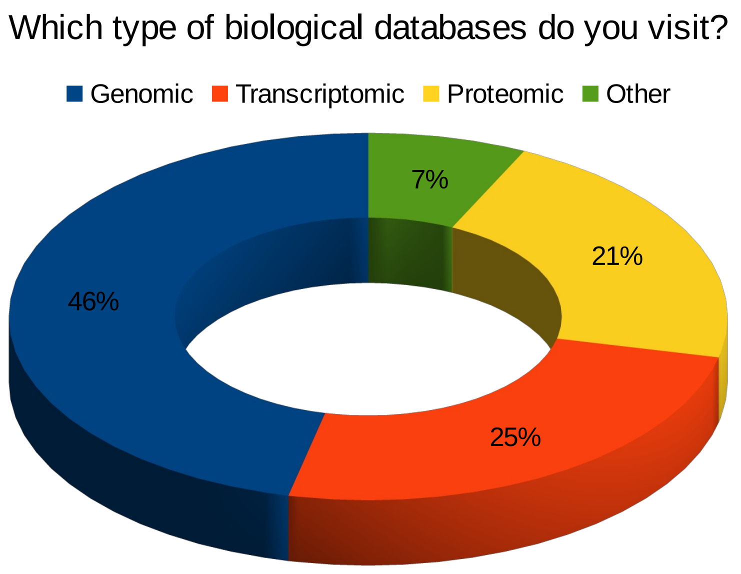


Figure 8: Spatial biological database user behavior

The charts on the previous page illustrate that more than half of the sampled population use biological databases at least once a week. Just less than half of the individuals use genomic databases. Two thirds of users indicated that they often have to traverse multiple biological databases to obtain information on a single biological feature, and that they know which specific database to query, but not how to use an API to query a database. When asked about waiting for results to be processed, 61% said that they do not mind waiting for a confirmation email to be sent once processing has been completed. 83% of the users sampled said that they were familiar with Pubmed Central's open access initiative (PMC-OAI) and 56% said that they use VCF files in their research. Given the option to choose more than one preferred method of being assisted when using a new database, a preference for video tutorials emerged, followed by user manuals, trial-and-error usage and online forums. Personal assistance ranked lowest of the preferred training methods. Real-time results of the survey can be viewed at https://goo.gl/3hWqIL.

Data extraction and database creation

The PMC-OAI corpus was provided in four batches with an average size of 4.2 Gb. Downloading could be completed overnight using a connection speed of 30 Mbps over a 802.11N wireless router. Scanning the downloaded article set revealed that 1.82% of the articles contained at least one probable reference SNP identifier (using the search criteria defined in the methods and materials section). Discarding articles that did not include at least one refSNP, reduced the uncompressed corpus size by a factor of 37. The following table (Table 1) elucidates this reduction per batch:

Table 1: Pubmed Central open access initiative corpus size

Batch name range	Articles in batch	Compressed size (Mb)	Unzipped size (Mb)	Articles containing refSNPs	Parsed size (Mb)
A-B	242 382	3 528	16 600	4 494	404
C-H	215 999	3 164	14 900	3 982	457
I-N	354 989	4 462	20 200	4 474	439
O-Z	318 714	5 651	26 300	7 700	800
	1 132 084	**16 805**	**78 000**	**20 650**	**2 100**

Analysis of the average character length of reference SNP identifiers extracted from the corpus (Figure 9) revealed that refSNPs within the PMC-OAI contained on average nine alphanumeric characters (the first two alphabetic characters followed by digits). Trimming of refSNPs containing less than six characters, to avoid ambiguity caused by unrelated naming conventions, were therefore justified.

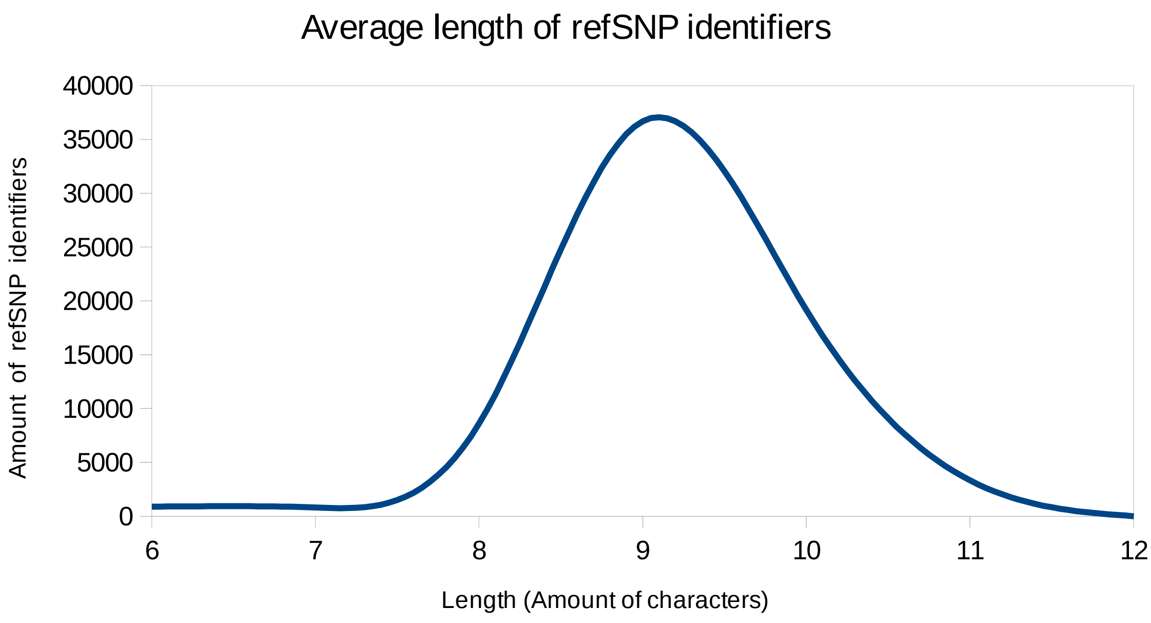


Figure 9: Average length of extracted refSNPs within the Pubmed Central open access initiative

A total of 69,463 unique refSNP identifiers and 8,743 unique (but case-sensitive) author defined keywords/phrases were left over after extraction and trimming. Further parsing of the subset to extract only relevant information required for each refSNP, reduced the size of the data eventually stored within the `JSONdbKeys.json` database to 58.4 Mb. The following snippet illustrates a raw entry from the resulting database:

```
{"rs_number": "rs12940887",
"email_address": "vinit.sawhney@bartshealth.nhs.uk",
"publication_date": "2012",
"pubmed_id": "3426779",
"doi": "10.2174/138920212802510484",
"pubmed_file_name": "Curr_Genomics_2012_Sep_13(6)_446-462.nxml",
"rs_number_cited_in_article": "2",
"article_title": "Current Genomics in Cardiovascular Medicine",
"keywords": ["Cardiovascular disease", "GWAS", "Gene sequencing",
"Personalised medicine."]}
```

Web application user interface

SNiPhunter was hosted on the SANBI mainframe and assigned the sub-domain name sniphunter. The complete URL for the search engine was http://sniphunter.sanbi.ac.za The index/main page of the SNiPhunter web application, as imaged in Figure 10, contained the SNiPhunter brand name in the top left hand corner. A trigram in the top right hand corner opened a drop-down list (Figure 11) when clicked. The drop-down list contained links to all the features offered by the web application:

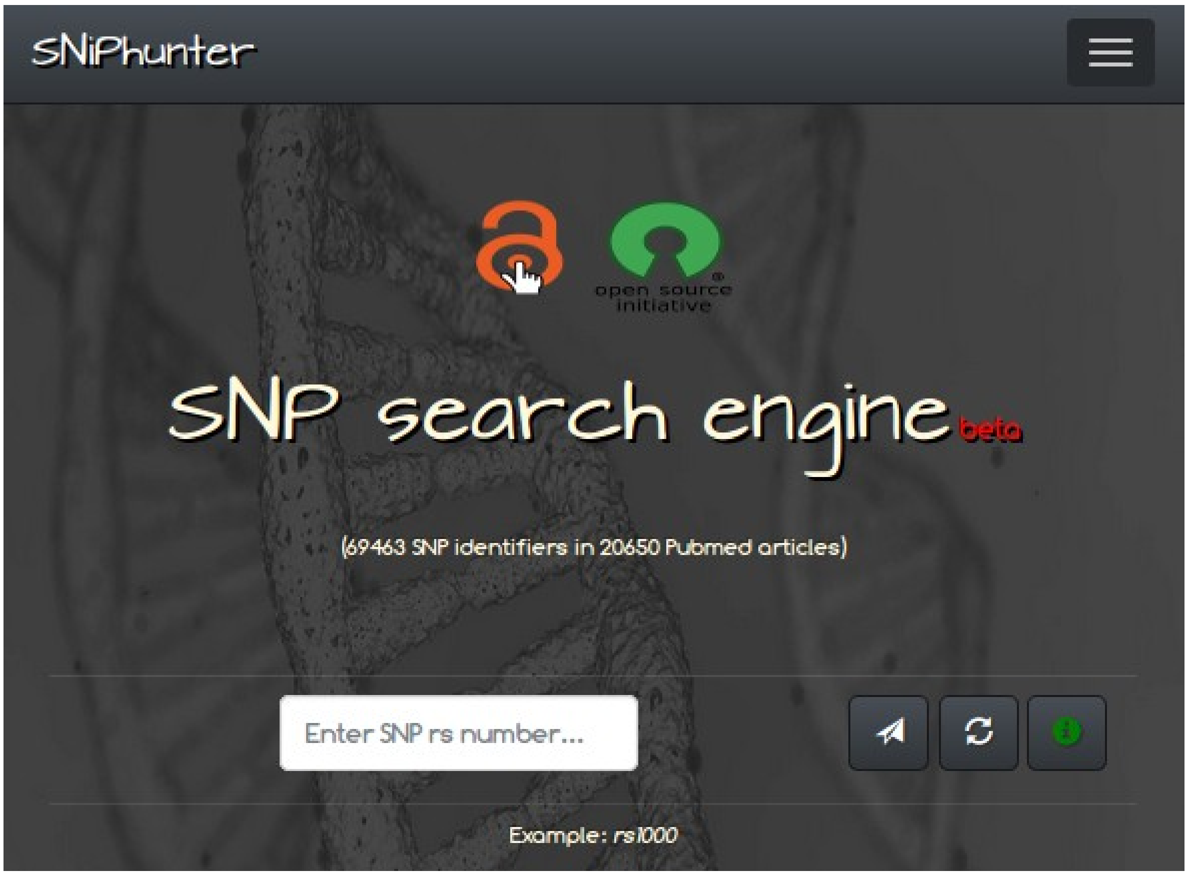


Figure 10: Main page of the SNiPhunter SNP search engine

Clicking the lock image below the navigation bar opened a link to PLOS magazine's description of open access initiatives (https://www.plos.org/open-access/), while clicking the open source initiative logo led to the official website (https://opensource.org/) of the open source initiative. The main heading describing SNiPhunter as a SNP search engine, as well as a sub-heading with database statistics, followed below the image links. The interactive part of the index page contained an input field, which would open a predictive text feature (Figure 12) once brought into focus. To the right of the input field was a query launch button (paper plane icon), a refresh button (circular arrows icon), and an information button (green info icon) that opened an info modal (Figure 13) when clicked. The index page and all other pages were responsive and the content of the web application adjusted satisfactory (using http://mobiletest.me) to all emulated devices.

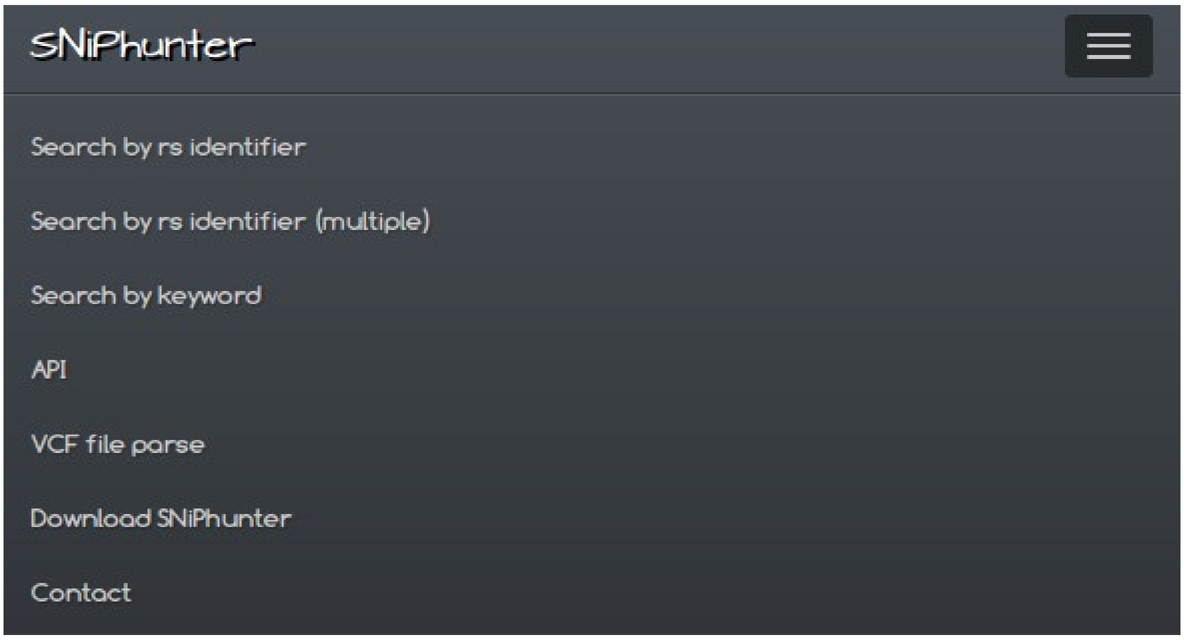
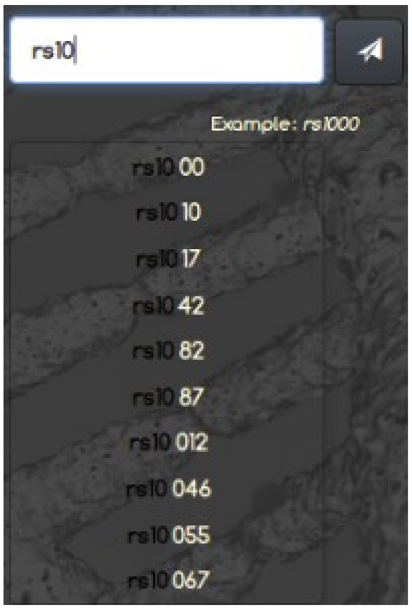
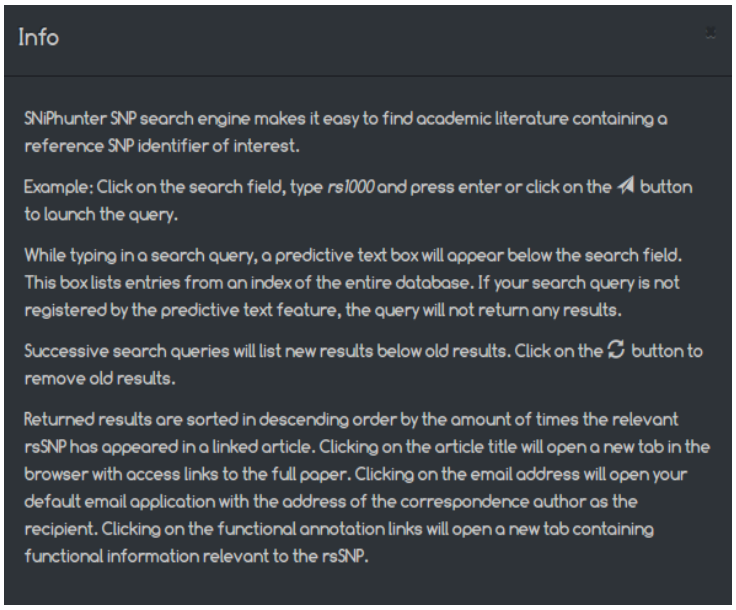


Figure 11: Trigram drop-down displaying link to all website features





Info

SNiPhunter SNP search engine makes it easy to find academic literature containing a reference SNP identifier of interest.

Example: Click on the search field, type *rs1000* and press enter or click on the button to launch the query.

While typing in a search query, a predictive text box will appear below the search field. This box lists entries from an index of the entire database. If your search query is not registered by the predictive text feature, the query will not return any results.

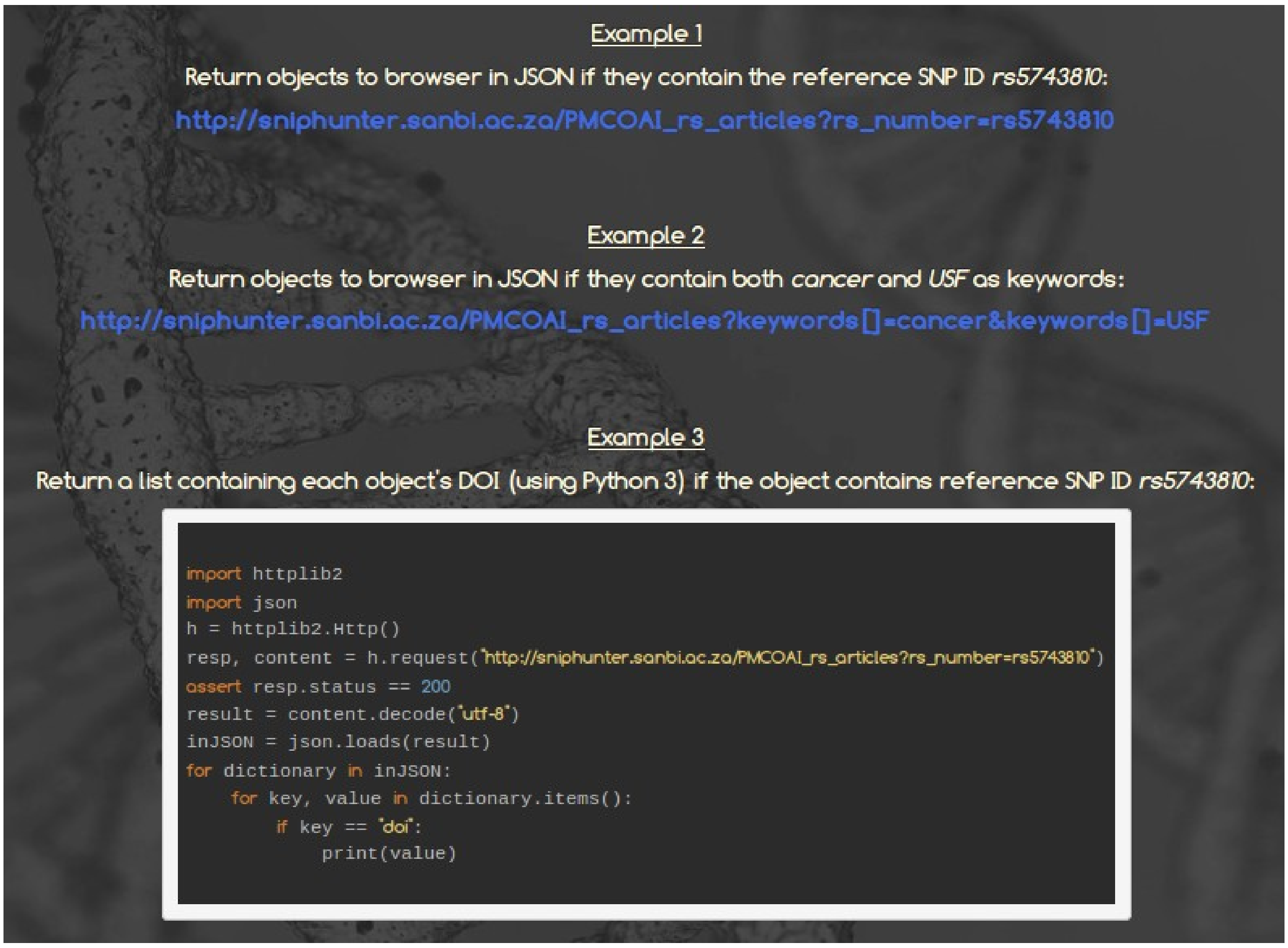
Successive search queries will list new results below old results. Click on the button to remove old results.

Returned results are sorted in descending order by the amount of times the relevant rsSNP has appeared in a linked article. Clicking on the article title will open a new tab in the browser with access links to the full paper. Clicking on the email address will open your default email application with the address of the correspondence author as the recipient. Clicking on the functional annotation links will open a new tab containing functional information relevant to the rsSNP.

Figure 12: Predictive text feature

Figure 13: Info modal content

The page offering keyword search functionality had a similar layout to the refSNP search page, except that there was a link that took the user to the API page if multiple keyword search queries were to be made. The API page contained three examples (Figure 14) to familiarize the user with using the API provided by the SNiPhunter web service. The first example illustrated the making of a single query, the second illustrated the making of multiple queries, and the third example illustrated the use of Python in simultaneously making and filtering queries.



Example 1

Return objects to browser in JSON if they contain the reference SNP ID *rs5743810*:

http://sniphunter.sanbi.ac.za/PMCOAI_rs_articles?rs_number=rs5743810

Example 2

Return objects to browser in JSON if they contain both *cancer* and *USF* as keywords:

http://sniphunter.sanbi.ac.za/PMCOAI_rs_articles?keywords[]=cancer&keywords[]=USF

Example 3

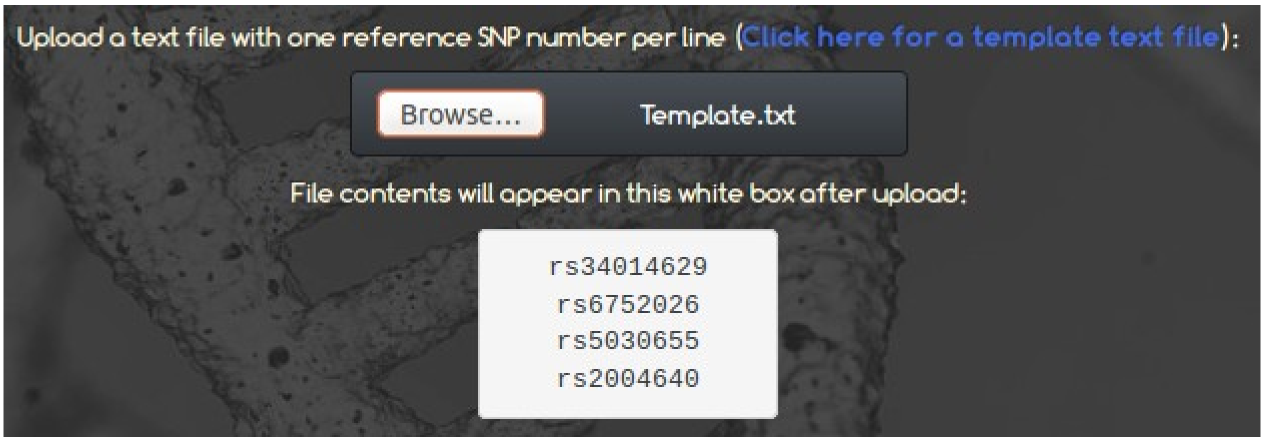
Return a list containing each object's DOI (using Python 3) if the object contains reference SNP ID *rs5743810*:

```
import httplib2
import json
h = httplib2.Http()
resp, content = h.request("http://sniphunter.sanbi.ac.za/PMCOAI_rs_articles?rs_number=rs5743810")
assert resp.status == 200
result = content.decode("utf-8")
inJSON = json.loads(result)
for dictionary in inJSON:
    for key, value in dictionary.items():
        if key == "doi":
            print(value)
```

Figure 14: Examples illustrating the use of SNiPhunter's API

The third example in the API could be amended by replacing the term *doi* in the second from last line with *email_address*, *publication_date*, *pubmed_id*, *pubmed_file_name*, *keywords* or *rs_number_cited_in_article* depending on what information the user required.

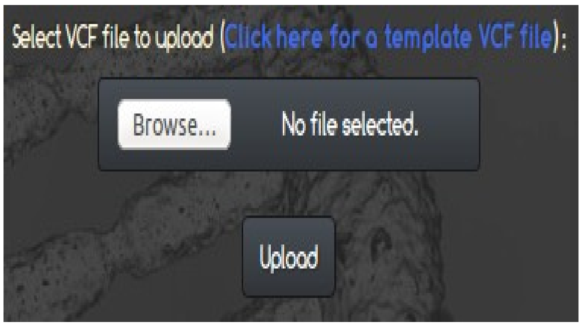
Since queries using multiple refSNPs often contain more terms than can easily be included in a search query string, an upload feature was provided on the multiple refSNP search page (Figure 15). Uploaded content was correctly displayed as indicated by the listing of submitted refSNPs in the white box at the bottom of Figure 15. To assist the user in creating text files for submission to the search engine, a VCF file parser was made available (Figure 16) that could (i) extract rs numbers from a VCF file and (ii) extract genome co-ordinates from a VCF file. Upload confirmation for VCF files were provided on a separate page (Figure 17). This page also contained parsing option. Both the VCF file parsing page and the multiple refSNP search page, contained a link to a template file that exemplified the required format of upload file contents.


Upload a text file with one reference SNP number per line (Click here for a template text file):

File contents will appear in this white box after upload:

```
rs34014629
rs6752026
rs5030655
rs2004640
```

Figure 15: Multiple refSNP search feature with upload confirmation


Select VCF file to upload (Click here for a template VCF file):

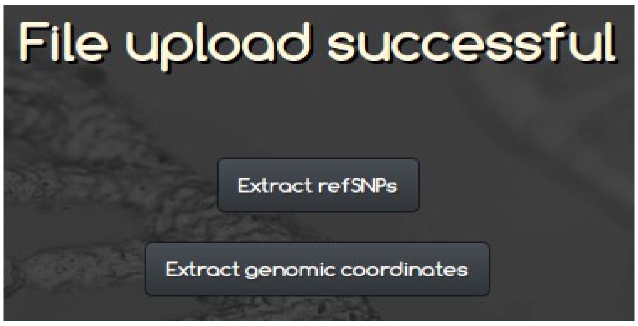
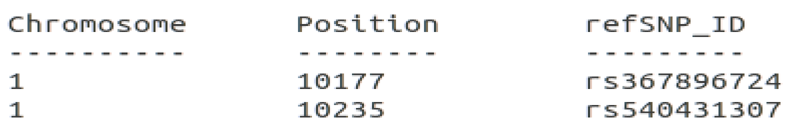

File upload successful

Figure 16: VCF file upload form

Figure 17: VCF file upload confirmation with parsing options

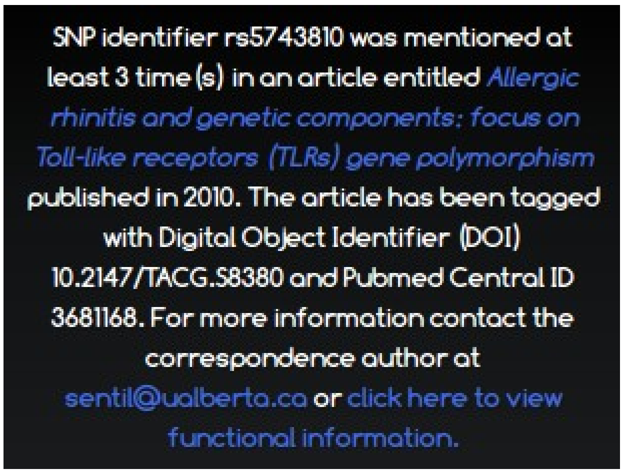
Extracted refSNPs were displayed as expected with one refSNP per line (without a heading) to comply with the search engine's submission format, while genome co-ordinates were displayed with headings (Figure 18):



```
Chromosome         Position         refSNP_ID
----------         --------         ---------
1                  10177            rs367896724
1                  10235            rs540431307
```

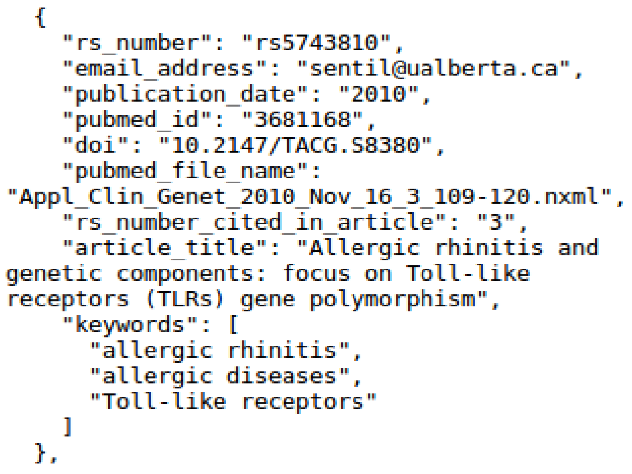
Figure 18: Sample genome co-ordinate data from parsed VCF file

The following images illustrate a refSNP based query result returned to the browser with the use of the web application (Figure 19) and the web service (Figure 20):



SNP identifier rs5743810 was mentioned at least 3 time(s) in an article entitled *Allergic rhinitis and genetic components: focus on Toll-like receptors (TLRs) gene polymorphism* published in 2010. The article has been tagged with Digital Object Identifier (DOI) 10.2147/TACG.S8380 and Pubmed Central ID 3681168. For more information contact the correspondence author at sentil@ualberta.ca or click here to view functional information.

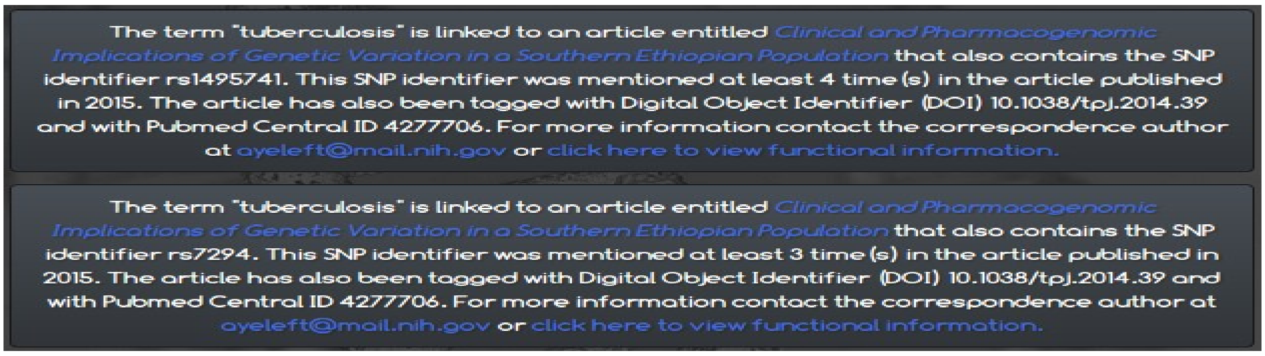
Figure 19: Typical web application result for refSNP based searches



```
  {
    "rs_number": "rs5743810",
    "email_address": "sentil@ualberta.ca",
    "publication_date": "2010",
    "pubmed_id": "3681168",
    "doi": "10.2147/TACG.S8380",
    "pubmed_file_name":
"Appl_Clin_Genet_2010_Nov_16_3_109-120.nxml",
    "rs_number_cited_in_article": "3",
    "article_title": "Allergic rhinitis and
genetic components: focus on Toll-like
receptors (TLRs) gene polymorphism",
    "keywords": [
      "allergic rhinitis",
      "allergic diseases",
      "Toll-like receptors"
    ]
  },
```

Figure 20: Typical web service result for refSNP and keyword based searches

Keyword based query results were similar to results returned for refSNP queries when using the web service, but were differently formatted for display to the user (Figure 21) when using the web application:



The term "tuberculosis" is linked to an article entitled *Clinical and Pharmacogenomic Implications of Genetic Variation in a Southern Ethiopian Population* that also contains the SNP identifier rs1495741. This SNP identifier was mentioned at least 4 time(s) in the article published in 2015. The article has also been tagged with Digital Object Identifier (DOI) 10.1038/tpj.2014.39 and with Pubmed Central ID 4277706. For more information contact the correspondence author at ayeleft@mail.nih.gov or click here to view functional information.

The term "tuberculosis" is linked to an article entitled *Clinical and Pharmacogenomic Implications of Genetic Variation in a Southern Ethiopian Population* that also contains the SNP identifier rs7294. This SNP identifier was mentioned at least 3 time(s) in the article published in 2015. The article has also been tagged with Digital Object Identifier (DOI) 10.1038/tpj.2014.39 and with Pubmed Central ID 4277706. For more information contact the correspondence author at ayeleft@mail.nih.gov or click here to view functional information.

Figure 21: Keyword based query results illustrating that results are sorted by the amount of times refSNPs occur in the associated article

Figure 21 also illustrates a built-in sort functionality that returned results to the user based on the amount of times a refSNP occurred within associated articles. Articles more relevant to a certain refSNP (based on the assumption that occurrence frequency determines relevance) were listed first in both refSNP and keyword based query results.

Query response time

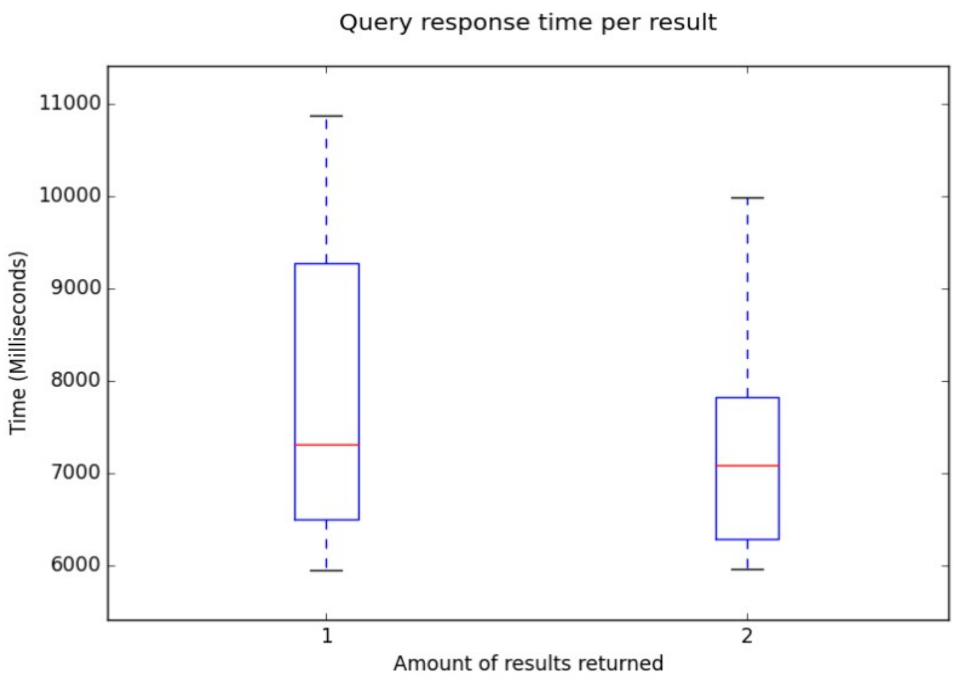


Figure 22: Time taken to return single and multiple results for a single query

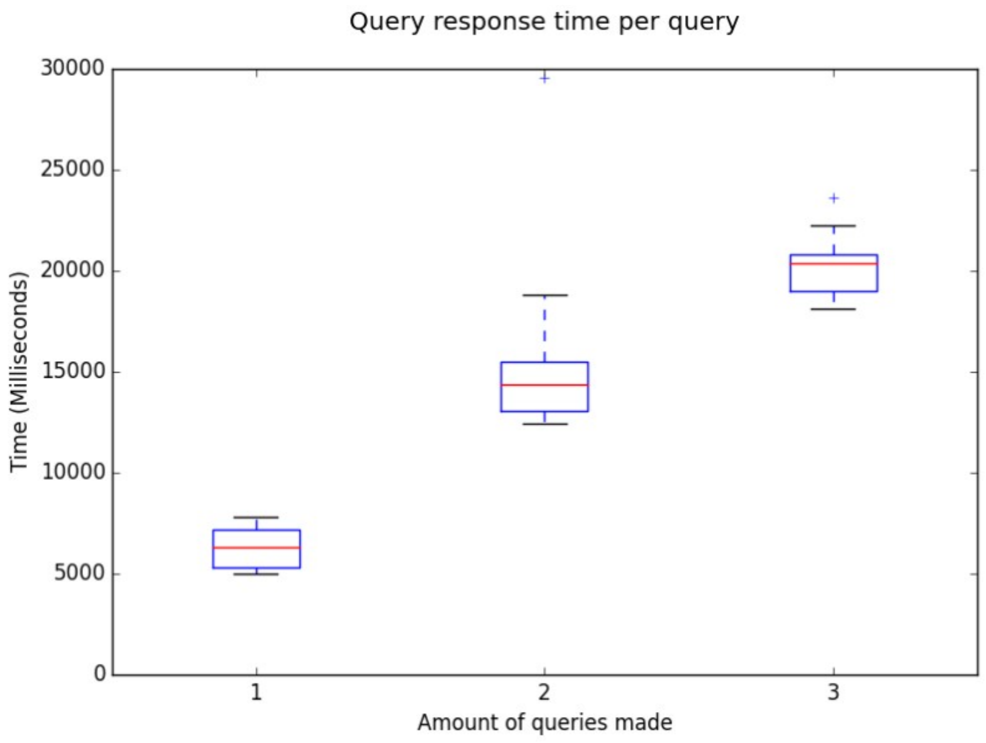


Figure 23: Time taken to return results with incremental queries

There was no apparent difference between the time taken to return single vs multiple results on a single query (Figure 22) with the average time taken to return results being about seven and a half seconds. However, when using the multiple search feature, query response time did increase with each additional query launched (Figure 23). Regression analysis revealed that the average time in milliseconds to return results increased according to the following formula:

$$F(x)=7026x+165$$

Database validation

Using four randomly selected refSNPs and three randomly selected keywords/phrases to test the validity of results retrieved from the database confirmed (Table 2) that information in the database was correctly stored, retrieved and displayed to the user:

Table 2: Web application and service test using random terms

Identifier, keyword or phrase	**Appear in predictive text**	**Correct internal results**	**Correct external results**	**API returns data object to browser**	**API accessed with python**	**No results returned**
rs8702	✔	✔	✔	✔	✔	
rs4323662	✔	✔	✔	✔	✔	
rs1219109046	✔	✔	✔	✔	✔	
rs00000				✔	✔	✔
tuberculosis	✔	✔	✔	✔	✔	
mouse mutant	✔	✔	✔	✔	✔	
unkeyword				✔	✔	✔
<empty string>						✔

Client side performance measurement

Google PageSpeed Insights reported that website performance could be improved by gzip resource compression, however, since it also reported that server response time was adequate and that there were unforeseen consequences with implementing a Node compression package, the suggested change was postponed. Other suggested changes by PageSpeed Insights had a trivial impact on speed and user experience and were ignored. Supplementary results from WebPageTest revealed that the load time for the SNiPhunter website was 5926 ms (Figure 24) and re-load time was 1910 ms (Figure 25):

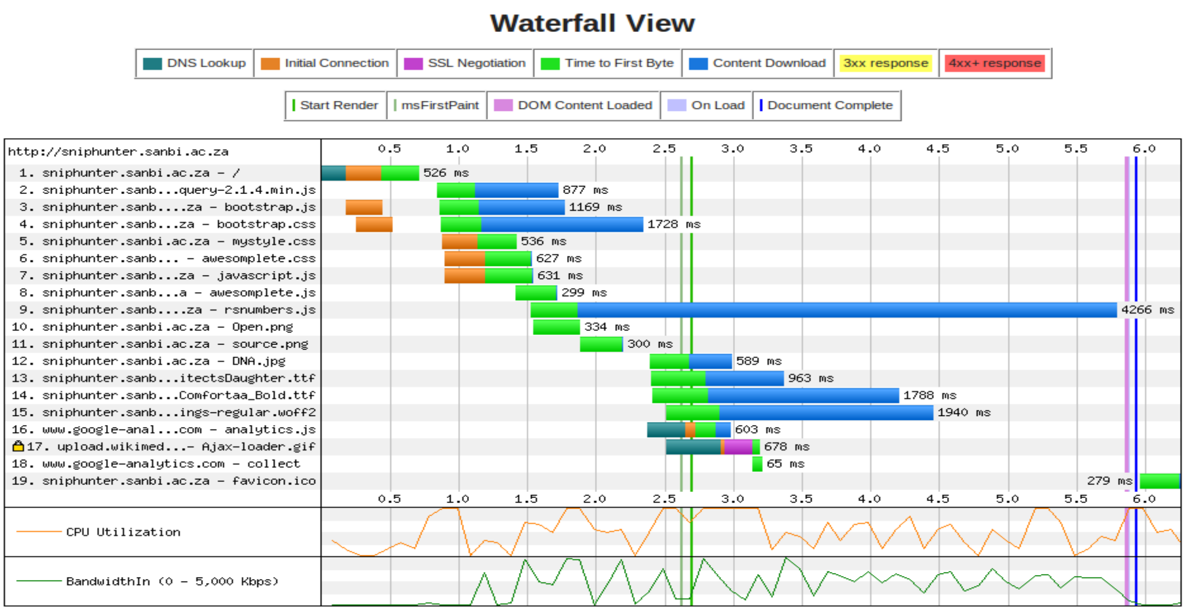


Figure 24: SNiPhunter index page load time according to WebPageTest

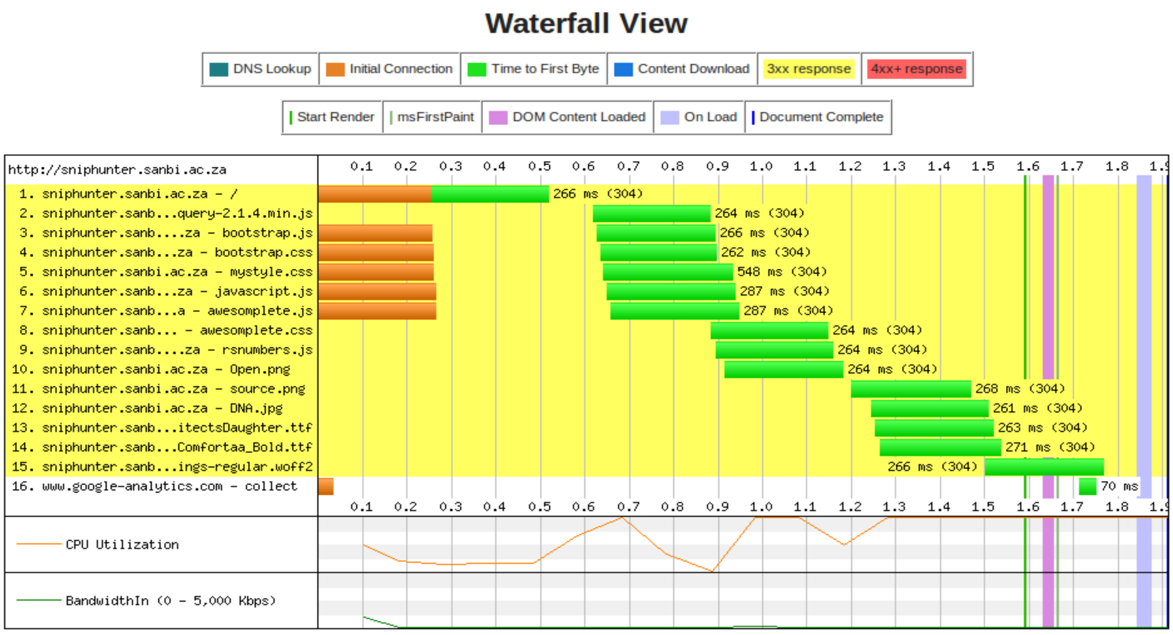


Figure 25: SNiPhunter index page re-load time according to WebPageTest

The bulk of the 5926 ms during first-load time was taken up by the download of *rsnumbers.js* (the file containing refSNPs for the predictive text feature). Page rendering was however not prevented by this file since it was called at the very end of the body tag within the HTML of the page.

User participation analysis

Google analytics for the month of January 2016 reported 484 visitors to the site of which 230 users (47.31%) were unique. An average of 1.67 pages were viewed per visit with the average duration of a visit lasting four minutes and twenty-five seconds. 27.27% of the visitors viewed more than one page. Similar results for December 2015 showed 194 visits of which 188 were unique (95.36%), with 1.15 pages viewed per visit. Average visit duration was twenty-four seconds, while 5.15% of the visitors viewed more than one page.

Post-implementation survey

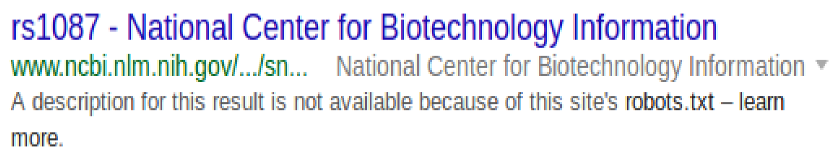
A survey carried out to gather user input on SNiPhunter's beta release indicated that most users viewed the site using the Firefox browser and the rest used the Chrome browser. The web application's design appeal (the look and feel of the website) received an average score of eight out of ten. 85.7% of those surveyed reported that results returned using the refSNP search feature was useful. No user indicated that a feature of the website was too complex, although most users did not complete testing all of the features. Real time results can be viewed at https://goo.gl/ccGgwb.

DISCUSSION

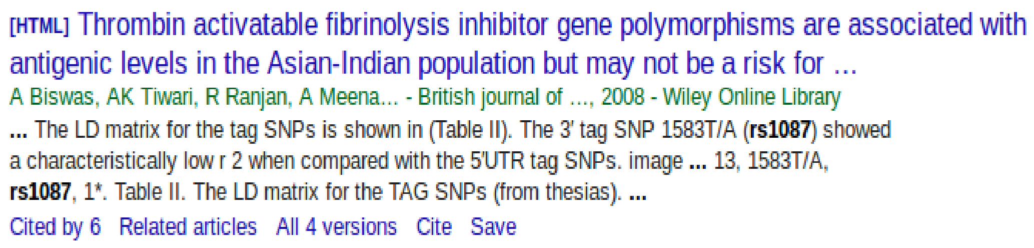
Comparing SNiPhunter results to those of other search engines

The following four images (Figure 26, A-D) compare the first result retrieved when searching for refSNP related academic literature using Google, Google Scholar, dbSNP and SNiPhunter:

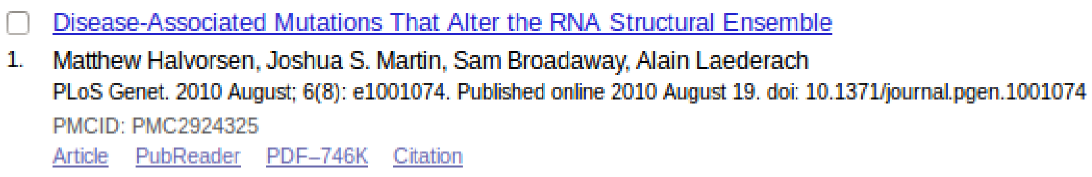
A


rs1087 - National Center for Biotechnology Information
www.ncbi.nlm.nih.gov/.../sn... National Center for Biotechnology Information
A description for this result is not available because of this site's robots.txt – learn more.

B


[HTML] Thrombin activatable fibrinolysis inhibitor gene polymorphisms are associated with antigenic levels in the Asian-Indian population but may not be a risk for ...
A Biswas, AK Tiwari, R Ranjan, A Meena... - British journal of ..., 2008 - Wiley Online Library
... The LD matrix for the tag SNPs is shown in (Table II). The 3′ tag SNP 1583T/A (**rs1087**) showed a characteristically low r 2 when compared with the 5′UTR tag SNPs. image ... 13, 1583T/A, **rs1087**, 1*. Table II. The LD matrix for the TAG SNPs (from thesias). ...
Cited by 6 Related articles All 4 versions Cite Save

C


Disease-Associated Mutations That Alter the RNA Structural Ensemble
1. Matthew Halvorsen, Joshua S. Martin, Sam Broadaway, Alain Laederach
PLoS Genet. 2010 August; 6(8): e1001074. Published online 2010 August 19. doi: 10.1371/journal.pgen.1001074
PMCID: PMC2924325
Article PubReader PDF–746K Citation

D

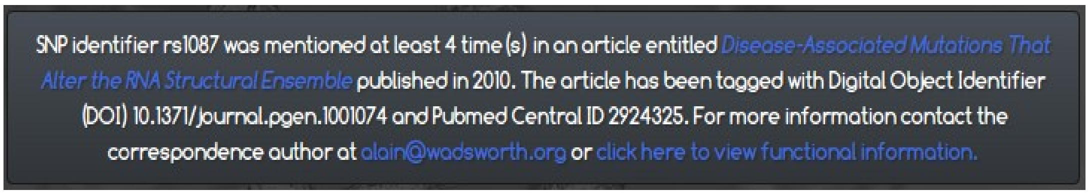

SNP identifier rs1087 was mentioned at least 4 time (s) in an article entitled *Disease-Associated Mutations That Alter the RNA Structural Ensemble* published in 2010. The article has been tagged with Digital Object Identifier (DOI) 10.1371/journal.pgen.1001074 and Pubmed Central ID 2924325. For more information contact the correspondence author at alain@wadsworth.org or click here to view functional information.

Figure 26: Comparison of SNiPhunter results with other search engine results: (**A**) Google search (**B**) Google Scholar (**C**) Pubmed Central (**D**) SNiPhunter

Major search engines, such as Google, design their search algorithms to retrieve information on all subjects and this is reflected by the lack of specificity in their results (Figure 26, A – B). When searching for academic literature, it is therefore necessary to use subject specific search engines. Using the search engine feature provided by PMC (the database from which the dataset for SNiPhunter was derived) returns the same first result as a SNiPhunter search, but two features of the SNiPhunter search engine makes its results more informative: (i) an indication is given to the user as to how many times a refSNP has appeared in the suggested article and (ii) the email address of the contact author is provided within the results. With these additional features (identified during the literature review as shortcomings in academic literature search engine results) the user is both able to determine the relevance of the suggested article quantitatively, and immediately gain access to the person who is responsible for answering questions on the suggested article. This second feature could be of particular use by saving time when multiple requests are made. In fact, the PMC search engine gives no indication as to how a user should launch multiple refSNP based queries simultaneously.

The benefits of using SNiPhunter to search for academic literature, when using keywords instead of refSNPs, sets it further apart from its competition. Since queries are made to the SNiPhunter database on the condition that each database entry has a reference to at least one refSNP, the user is guaranteed to get results that make reference to scientific literature containing at least one refSNP.

Multiple queries and VCF parsing

Besides from the two novel features that SNiPhunter brings to academic literature retrieval, the web application also endeavors to make repetitive queries and VCF file parsing more user friendly, by removing the expectation that currently rests on the user to have some scripting knowledge. The web application that was developed during this project allows a user to upload and process files with the click of a button (with instructions and template files being provided on the relevant pages). This approach ensures that biological database users who do not have a background in programming do not get discouraged from using the database. For example, in the event that a user wishes to launch multiple queries, the only expectation is that the user knows how to create a text file, insert each refSNP on a new line, save the file onto the local disc, and submit it using the online submission form. There are currently no file size upload limits in place for the multiple query feature, but as pointed out in the methods and materials, the file is not actually uploaded to the server. The limitation on processing capacity would therefore depend on the user's device. According to the regression function, a file containing ten refSNPs would typically take just over a minute to process. The limiting factor would accordingly be processing time rather than processing capacity. VCF files in contrast are much larger than text files, so processing capacity becomes are real concern. SNiPhunter currently imposes no restrictions on VCF file size uploads. Unlike with the text files used during multiple queries, VCF files are uploaded to the server. To avoid the server being overloaded with large VCF files, the temporary directory for file uploads are cleared on a daily basis. It will be necessary for the user to reduce VCF files sizes to manageable sizes (about 1Mb is recommended), rather than place a burden on the server to handle these potentially huge files. External data manipulation tools such as the 1000 genomes data slicer (http://www.1000genomes.org/data-slicer) or tools available on the Galaxy server would have to be used prior to uploading a VCF file for parsing. This could be considered a shortcoming in the parsing feature, but processing massive VCF files on the SANBI mainframe is currently not an option.

SNiPhunter ontology

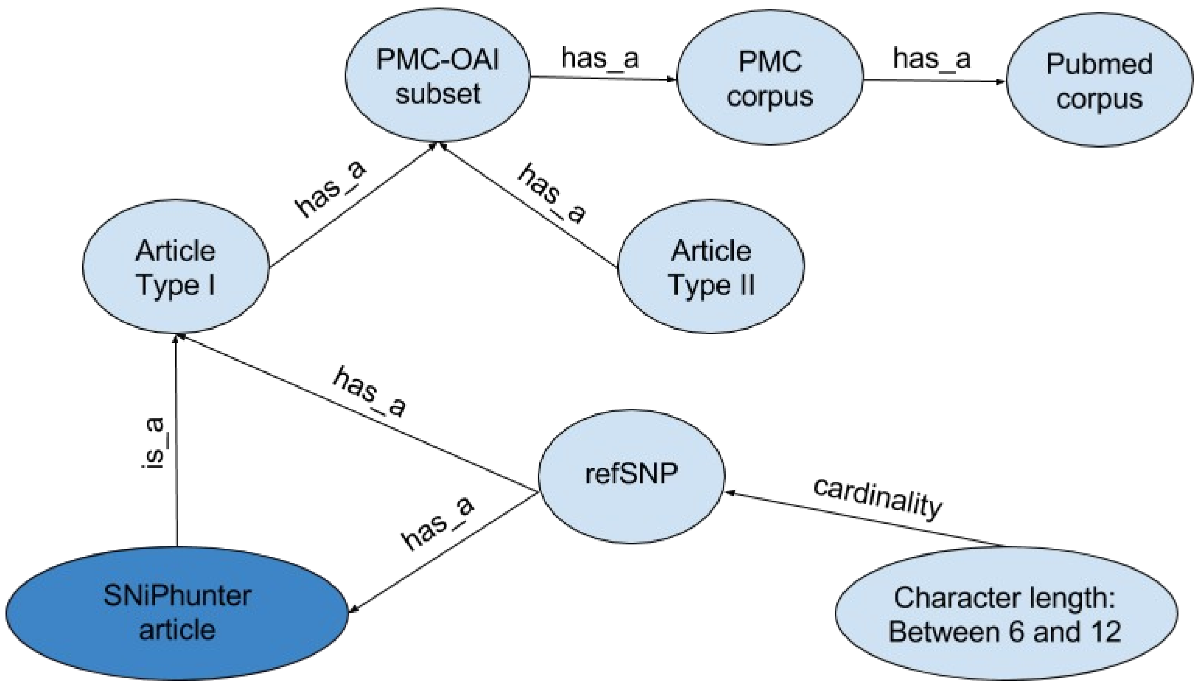


Figure 27: Ontological diagram depicting SNiPhunter's article membership to the Pubmed corpus

The preceding ontological diagram (Figure 27) defines a SNiPhunter article in terms of its Pubmed membership. According to Pubmed (https://www.ncbi.nlm.nih.gov/pubmed), their database contains approximately 25 million indexed citations to life sciences journals and online books. Articles within Pubmed Central (a subdivision of Pubmed), contain about 3.7 million full free-text articles (https://www.ncbi.nlm.nih.gov/pmc/). Yet another subset of the PMC subdivision, is the PMC-OAI subset, which is made available to the public for data mining related exercises. PMC corpus extraction results show the volume of this subset to be about 1.1 million. SNiPhunter's corpus is a third subset that contains only articles mentioning at least one refSNP (if the refSNP has a character count of between six and twelve). Currently this refSNP subset contains about twenty thousand articles.

SNiPhunter use case

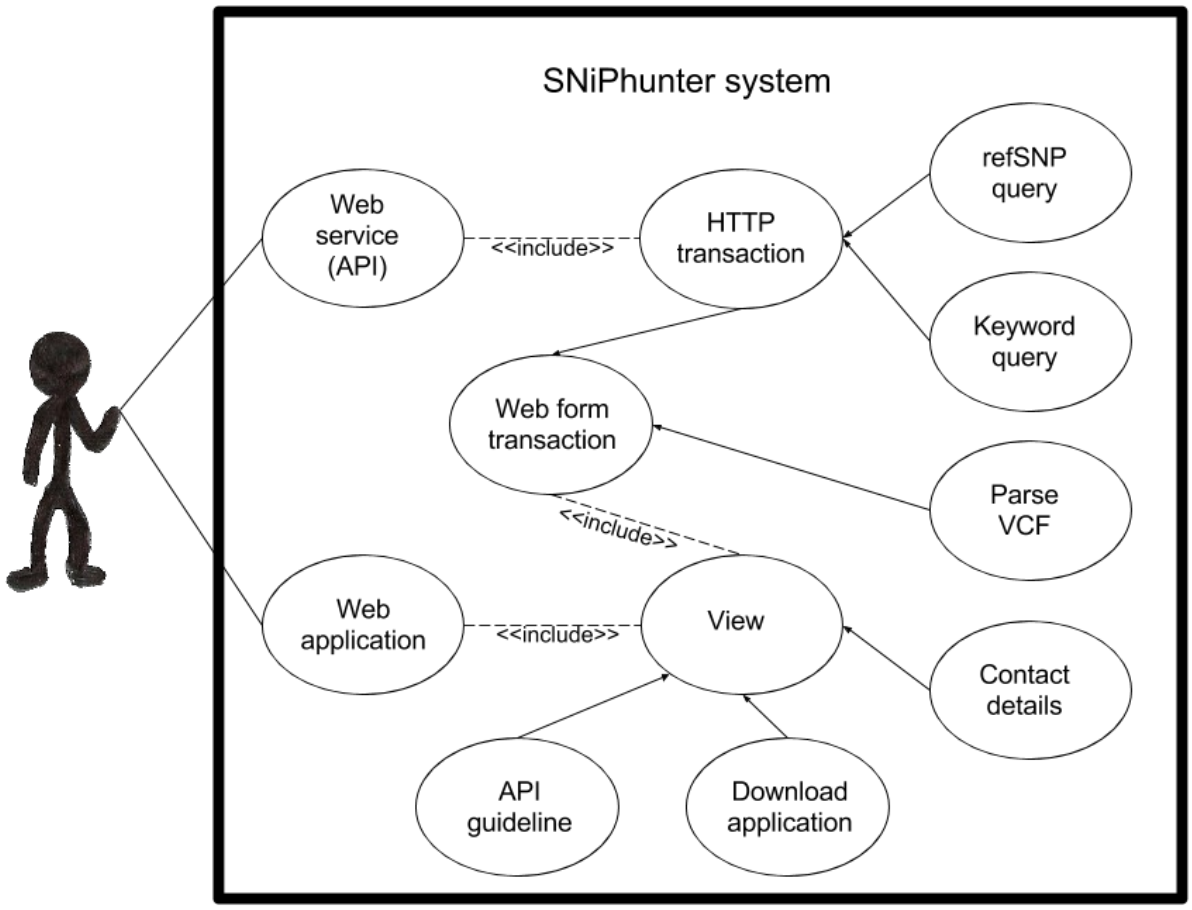


Figure 28: SNiPhunter use case diagram

SNiPhunter's use case diagram (Figure 28) highlights its dual interaction model: a user can query the database using either the web application or the web service. The diagram further illustrates that all queries to the database are RESTful since queries launched using the web application leads to programmatic HTTP requests. The web application can be accessed by simply opening a browser and navigating to http://sniphunter.sanbi.ac.za.

Calls to the API are made in a similar manner but a term indicating that the call must be routed to the database is suffixed to the URL followed by a separator and a term indicating whether a search should be conducted by refSNP or by keyword. When using a scripting language, the returned objects can be filtered for specific data. Examples one through three on the API guideline page (http://sniphunter.sanbi.ac.za/API.html) show two possible query strings and a scripted call. Query strings should always be constructed to retrieve objects by supplying a valid rs number (refSNP) or keyword since the API will only return an empty object if the database does not contain data on the query.

User behavior and feedback

User behavior was analyzed using both Google analytics and in-house surveying. This was of benefit to the project since Google analytics gave an external view on user behavior, while the in-house survey was conducted by targeting specific users at the Natural Sciences faculty of the University of the Western Cape. A map (Figure 29) of the geographic dispersion of users indicated that SNiPhunter was accessed from around the globe with most interaction coming from Russia and the United States:

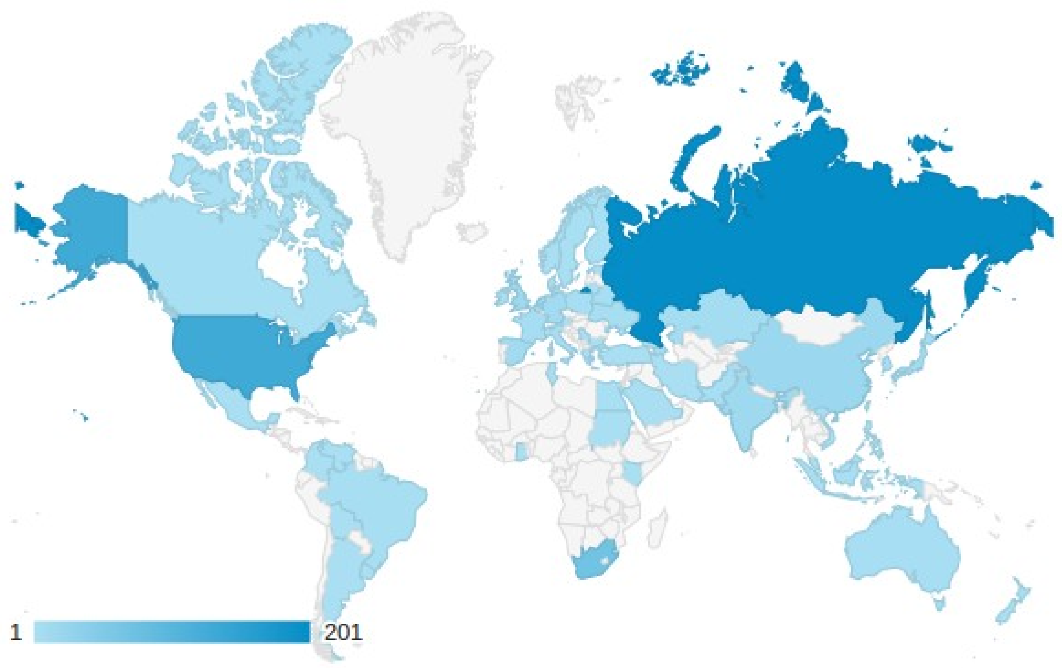


Figure 29: Geographic dispersion of SNiPhunter users for the period December 2015 to January 2016

This dispersion supported the argument in the literature review in that providing a web application, with a user-friendly interface for querying scientific databases, broadens the target audience. Incremental saturation in Figure 29 indicates the amount of sessions that were initiated with the SNiPhunter web application and excludes calls to the API. It is important to note that the data that was used to generate this map included sessions from users who might not be genuine scientific users. It is a reflection of all interaction with the website. The session duration and geographic dispersions on national levels, however, suggested that sessions were not initiated from single origins within each country and that the users spent at least some time on the website. This validated the authenticity of the data to an extent. For example, a similar map of sessions originating just from within South Africa (Figure 30), reports activity from four provinces:

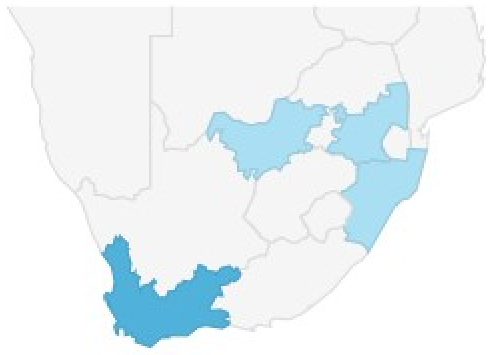


Figure 30: User sessions originating from South Africa for the period December 2015 to January 2016

The interaction from the province of Limpopo could be traced to a class exercise that was conducted at the University of Limpopo. Google analytics reported 17 unique visits from the city of Polokwane where the university is located. A contact person at the university confirmed that a group of 4th year Medical Sciences students used the SNiPhunter refSNP search feature to source articles during a Human Genetics exercise. This was the first proof of SNiPhunter being used in practice and revealed that these scientific users spent an average of three minutes on the site. A quarter of the visitors from this cohort explored additional features on the website, which might be an indication of interest.

The reported increase in visits to SNiPhunter's website from December 2015 to January 2016 revealed that the user base is growing. This is supported by an increase in visitors who viewed more than one page. Moreover, a drop in unique visitors suggested that users are being retained. The in-house surveys revealed that most of those surveyed used genomic databases frequently. The software developed during this project sought to cater for this audience and received an overall positive reception during the post implementation survey, however, one user complained about the lack of contrast in the color scheme of the website. The background color for dynamic content was accordingly changed from gray to red to accommodate users who might have difficulty reading the retrieved results. There was also an inquiry regarding the database not returning results for valid refSNPs. This issue is related to scalability and is discussed in more detail under the section dealing with the limitations of SNiPhunter.

Search engine optimization and semantic web integration

The structured vocabularies (HTML metatags, Dublin Core and RDFa), that were used to make the application developed during this project more accessible to web crawlers, seemed to have a greater impact on some search engines than others. Search engines tend to change their ranking criteria over time, and optimizing the structured data of a website to gain higher ranking, depends not only on what ranking criteria is used by a given search engine, but also whether structured data optimized for a given search engine would be effective with another search engine. A search using the term *SNiPhunter* with five popular search engines revealed that http://sniphunter.sanbi.ac.za was the top result on Google, Bing and Ask. However, SNiPhunter was ranked fifth on AOL search and ninth on Yahoo. The reason for the difference in ranking is not clear. Higher rankings might be seen with the two latter search engines in future if the size of the user base continues to grow. Google Webmaster Tools indicated that they had registered all the RDFa content of the website. In total, four RDFa compliant classes (http://schema.org) were discovered. They were: MedicalWebPage, SoftwareApplication, Person, and APIReference. These classes indicated the target audience, source code accessibility, the name of the developer, and the availability of an API, respectively.

Licensing and privacy

All the source files of this open source project is available on Github and is licensed under a Massachusetts Institute of Technology (MIT) license. This license was used because it is a permissive free software license that puts little restriction on reuse, which makes it ideal for minimizing license compatibility issues that might arise during mashup design. Moreover, many of the third party libraries used during this project used the MIT license.

The MIT license is worded as follow (italics added):

> *Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sub-license, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:*
>
> *The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.*
>
> *THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.*

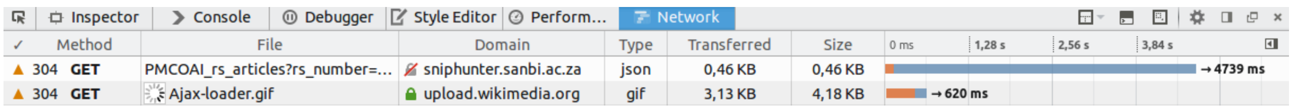
A file named COPYRIGHT NOTICE is available in the main directory of the source file repository on Github. It complies with the second paragraph of the MIT license by including the original MIT license for each third party library, and also lists the various other licenses under which non-MIT licensed third party packages were published. Due to the volume of life sciences articles available through the PMC-OAI, the individual licenses for each article could not be examined, but PMC's general copyright notice regarding use of PMC articles was included in the COPYRIGHT NOTICE file. It should further be noted from the PMC general copyright notice that articles published through PMC-OAI have creative commons or similar licenses, which tend to have lower reuse restrictions. Nevertheless, potential copyright issues were avoided by providing links to full text articles hosted on PMC instead of re-hosting relevant articles.

Since SNiPhunter's database only contains information that is already in the public domain, the only possible concern there could be regarding privacy in this project is whether correspondence authors have any objections to their email addresses being displayed prominently in search results. The spirit in which their email addresses are included in search results, however, justify the use: the intention is to promote communication between scientists. Moreover, the SNiPhunter database does not contain patient privileged, de-identified data or other information that could compromise the privacy of research subjects.

SNiPhunter limitations and future improvements

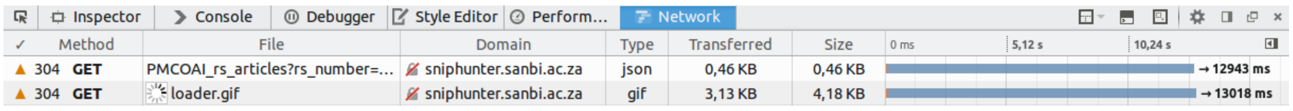
The first version of SNiPhunter, released on 29 January 2016, contained 69,463 refSNP identifiers and 8,743 keywords in 20,650 Pubmed articles. Although these statistics are highlighted to the user on the web application's front end, a concern was raised regarding SNiPhunter not returning any results for a given refSNP. A limiting feature of the database is that it only contains information made available to the public by the PubMed Central (PMC) open access initiative. This PMC subset contains much less literature than all of Pubmed's indexed literature (as illustrated with the ontological diagram). Nevertheless, it remains difficult to explain to a user why their reference SNP identifier or keyword query did not return any results. Some attempt is made in the information modals available on each of the query pages to inform the user that refSNPs or keywords that are not registered by the predictive text feature, are not indexed in the database. The ideal solution to the problem, however, is to incorporate data from multiple open access initiatives and automate updating of the data made available through these initiatives. Unfortunately, the investment of time and expertise required for additional implementation to meet this solution would require a group effort and is outside of the scope of a MSc project. Moreover, license restrictions on using automated downloaders are in place for the PMC-OAI corpus, which would prohibit automatic corpus updates. As mentioned in the literature review, the rate at which open access material is growing, is increasing over time. Taking this increase in material becoming available each year as well as the trend towards integrative data administration (also discussed in the literature review), there is a sense of optimism in the scientific community that tertiary data artifacts would in future be less hampered by heterogeneous distribution of scientific literature.

Since the third party code for SNiPhunter relied on software distributed free of charge, the first software package used in an attempt to meet a given implementation goal, did not always work. For example, a Node package that could be used to compress static web files, did compress the intended data, but interfered with the delivery of dynamic content generated by the JSON-server package. The respective first lines in the following two images compare the query load time before (Figure 31) and after (Figure 32) the compression package was imported:



✓	Method	File	Domain	Type	Transferred	Size	0 ms 1,28 s 2,56 s 3,84 s
▲ 304	GET	PMCOAI_rs_articles?rs_number=...	sniphunter.sanbi.ac.za	json	0,46 KB	0,46 KB	→ 4739 ms
▲ 304	GET	Ajax-loader.gif	upload.wikimedia.org	gif	3,13 KB	4,18 KB	→ 620 ms

Figure 31: Query response time without compression package (using external loading animation)



✓	Method	File	Domain	Type	Transferred	Size	0 ms 5,12 s 10,24 s
▲ 304	GET	PMCOAI_rs_articles?rs_number=...	sniphunter.sanbi.ac.za	json	0,46 KB	0,46 KB	→ 12943 ms
▲ 304	GET	loader.gif	sniphunter.sanbi.ac.za	gif	3,13 KB	4,18 KB	→ 13018 ms

Figure 32: Query response time with compression package (using local loading animation)

Figure 32 shows a query response time of 12943ms, which is well above the upper most deviance registered during query response time testing. In addition, the respective second lines in the above two images reveal that asynchronous loading is preferred since increased loading times will not prevent subsequent calls from executing successfully. It was for this reason that the animation indicating that an Ajax call is in progress was loaded from an external source rather than making a separate call to the same address to where an in-progress Ajax call was being made.

Seeing that open source software does not come with the same type of support that might be expected from commercial software, resolving issues encountered during usage depends on the goodwill and availability of the developer. There seemed to be a qualitative correlation between the robustness of a third party library and the developer answering queries in a timely and helpful manner. This makes sense given that a software package that has been abandoned would likely have compatibility issues. A good rule of thumb is to view the public activity thread of a repository to ascertain whether there is any active development taking place. Nevertheless, packages showing little or no active development could still be highly useful. The WebAwk wrapper for Node used during this project is an example of such a package.

CONCLUSION

A software solution called SNiPhunter was developed after a gap in scientific literature sourcing was identified during a literature review. Specific shortcomings noted during the review were (i) the inability to search for scientific literature using a quantitative estimate of the amount of times a refSNP appears in an article, and (ii) the inability to make such queries in bulk using a file uploaded via a user friendly interface. A subset of Pubmed Central's open access initiative was used to design a JSON flat-file database containing data necessary to operate an academic literature search engine as a web application and an API. The system was built from the ground up using only third party libraries and snippets from independent programmers to supplement the source code. No content management system, web design template, or commercial services were employed. Data extraction was carried out using the Python programming language, while data storage and retrieval were made possible using source code written for the Node interpreter. In addition to addressing the two observed shortcomings, SNiPhunter also offers a VCF parser integrated with the user interface to make using the system easier for scientist who do not have sufficient programming skills. Indeed, the entire approach to the project was user centric with an emphasis on using structured data and metadata vocabularies to increase the visibility of the application. A user survey was carried out prior to development and triangulated with a post development survey. These surveys together with website analytics data suggested that SNiPhunter was well received by the sampled population and that the user base was steadily growing. An ontological representation illustrated the constraints of the search engine and a distribution of refSNP character length was used to justify exclusion of ambiguous low character length entries. A use case diagram was constructed to highlight the dual interaction model of the developed software, and the source code of the project was published online under a MIT license, to encourage further development. The heterogeneity of scientific literature datasets was observed as a scalability obstacle for further development of SNiPhunter, but a trend in data integration technologies was recognized as a possible future solution to this problem. In summary, the creation of a viable academic search engine was demonstrated with the use of free material and tools.

REFERENCES

Abate, P., DiCosmo, R., Treinen, R., & Zacchiroli, S. (2011, June). MPM: a modular package manager. In *Proceedings of the 14th international ACM Sigsoft symposium on Component based software engineering* (pp. 179-188). ACM.

Aday, L. A., & Andersen, R. (1974). A framework for the study of access to medical care. *Health services research*, *9*(3), 208.

Alkoshman, M. M. (2015). Unified Modeling Language and Enhanced Entity Relationship: An Empirical Study. *International Journal of Database Theory and Application*, *8*(3), 215-227.

Antelman, K. (2004). Do open-access articles have a greater research impact? *College & research libraries*, *65*(5), 372-382.

Archambault, E., Amyot, D., Deschamps, P., Nicol, A., Provencher, F., Rebout, R., & Roberge, G. (2014). Proportion of open access papers published in peer-reviewed journals at the European and world levels 1996–2013. *Science-Metrix-European Commission*.

Aronson, J. K. (2005). Commentary: Open access publishing: too much oxygen? BMJ, 330(7494), 759.

Arzberger, P., Schroeder, P., Beaulieu, A., Bowker, G., Casey, K., Laaksonen, L., ... & Wouters, P. (2004). Promoting access to public research data for scientific, economic, and social development. Data Science Journal, 3, 135-152.

Attwood, T. K., Gisel, A., Bongcam-Rudloff, E., & Eriksson, N. E. (2011). *Concepts, historical milestones and the central place of bioinformatics in modern biology: a European perspective*. INTECH Open Access Publisher.

Australian Open Access Support Group (AOASG). (2015). Welcome to the AOASG. [Web log post] Retrieved from http://aoasg.org.au/

Bertino, E., Ooi, B. C., Yang, Y., & Deng, R. H. (2005, April). Privacy and ownership preserving of outsourced medical data. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on* (pp. 521-532). IEEE.

Bevacqua, N. (2015). *JavaScript Application Design*. Manning Publishing

Blumberg, R., & Atre, S. (2003). The problem with unstructured data. *DM REVIEW*, *13*(42-49), 62.

Bolser, D. M., Chibon, P. Y., Palopoli, N., Gong, S., Jacob, D., Del Angel, V. D., ... & Bhak, J. (2012). MetaBase—the wiki-database of biological databases. *Nucleic acids research*, *40*(D1), D1250-D1254.

Brickley, D., & Miller, L. (2012). FOAF vocabulary specification 0.98. *Namespace document*, *9*.

Brown, T.A. (2007). *Genomes 3.* New York, NY : Garland Science.

Buckley, J., Mens, T., Zenger, M., Rashid, A., & Kniesel, G. (2005). Towards a taxonomy of software change. *Journal of Software Maintenance and Evolution: Research and Practice*, *17*(5), 309-332.

Budapest Open Access Initiative (BOAI). (2002). Budapest open access initiative. Retrieved from http://www.budapestopenaccessinitiative.org/

Bush, V. (1945). As we may think. *Atlantic Monthly*, 101-108.

Cannataro, M., Guzzi, P. H., Tradigo, G., & Veltri, P. (2014). Biological Databases. Springer Handbook of Bio-/Neuroinformatics, 431-440.

Cappiello, C., Daniel, F., Matera, M., Picozzi, M., & Weiss, M. (2011). Enabling end user development through mashups: requirements, abstractions and innovation toolkits.

In *End-User Development* (pp. 9-24). Springer Berlin Heidelberg.

Chan, L., Kirsop, B., & Arunachalam, S. (2005). Open access archiving: the fast track to building research capacity in developing countries.

Cimino, J. J., Patel, V. L., & Kushniruk, A. W. (2002). The patient clinical information system (PatCIS): technical solutions for and experience with giving patients access to their electronic medical records. *International journal of medical informatics*, *68*(1), 113-127.

Cisco. (2014). Big Data: Not Just Big, But Different - Part 2 [Web log post] Retrieved from http://www.cisco.com/

Cock, P. J., Fields, C. J., Goto, N., Heuer, M. L., & Rice, P. M. (2010). The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic acids research*, *38*(6), 1767-1771.

Cui, B., Mei, H., & Ooi, B. C. (2014). Big data: the driver for innovation in databases. National Science Review, 1(1), 27-30.

Danecek, P., Auton, A., Abecasis, G., Albers, C. A., Banks, E., DePristo, M. A., ... & Durbin, R. (2011). The variant call format and VCFtools. *Bioinformatics*, *27*(15), 2156-2158.

Dearle, A. (2007). Software deployment, past, present and future. *Future of Software Engineering*, 269-284.

Demir, E., Cary, M. P., Paley, S., Fukuda, K., Lemer, C., Vastrik, I., ... & Finney, A. (2010). The BioPAX community standard for pathway data sharing. Nature biotechnology, 28(9), 935-942.

Di Battista, G., Didimo, W., Patrignani, M., & Pizzonia, M. (2002). Drawing database schemas. *Software: Practice and Experience*, *32*(11), 1065-1098.

Donabedian, A. (1972). Models for organizing the delivery of personal health services and criteria for evaluating them. *The Milbank Memorial Fund Quarterly*, 103-154.

Falagas, M. E., Pitsouni, E. I., Malietzis, G. A., & Pappas, G. (2008). Comparison of PubMed, Scopus, web of science, and Google scholar: strengths and weaknesses. *The FASEB journal*, *22*(2), 338-342.

Gansner, E. R., & North, S. C. (2000). An open graph visualization system and its applications to software engineering. *Software Practice and Experience*, *30*(11), 1203-1233.

Garfield, E. (2006). The history and meaning of the journal impact factor. *Jama*, *295*(1), 90-93.

Ghinita, G., Karras, P., Kalnis, P., & Mamoulis, N. (2007). Fast data anonymization with low information loss. In *Proceedings of the 33rd international conference on Very large data bases* (pp. 758-769). VLDB Endowment.

Gingeras, T. R., & Roberts, R. J. (1980). Steps toward computer analysis of nucleotide sequences. *Science*, *209*(4463), 1322-1328.

Goble, C., & Stevens, R. (2008). State of the nation in data integration for bioinformatics. *Journal of biomedical informatics*, *41*(5), 687-693.

Goodman, D. (2004). The criteria for open access. *Serials review*, *30*(4), 258-270.

Gulcher, J. R., Kristjansson, K., Gudbjartsson, H., & Stefansson, K. (2000). Protection of privacy by third-party encryption in genetic research in Iceland. *European journal of human genetics: EJHG*, *8*(10), 739-742.

Grote, C., Segreto, E., Okerlund, J., Kincaid, R., & Shaer, O. (2015, January). Eugenie: Multi-Touch and Tangible Interaction for Bio-Design. In Proceedings of the Ninth International Conference on Tangible, Embedded, and Embodied Interaction (pp. 217-224). ACM.

Handke, J. (2015). Micro-Lectures - Semantics. Retrieved from https://www.youtube.com/

Harnad, S., Brody, T., Vallieres, F., Carr, L., Hitchcock, S., Gingras, Y., ... & Hilf, E. R. (2008). The access/impact problem and the green and gold roads to open access: An update. Serials review, 34(1), 36-40.

Haug, C. (2013). The downside of open-access publishing. New England Journal of Medicine, 368(9), 791-793.

Hunter, K. (2005). Critical issues in the development of STM journal publishing. Learned publishing, 18(1), 51-55.

IBM. (2012). What is big data? Retrieved from http://www-01.ibm.com

Ives, B., & Olson, M. H. (1984). User involvement and MIS success: a review of research. Management science, 30(5), 586-603.

Janowicz, K., Hitzler, P., Adams, B., Kolas, D., & Vardeman, C. (2014). Five stars of Linked Data vocabulary use. *Semantic Web*, *5*(3), 173-176.

Kamphans, T., & Krawitz, P. M. (2012). GeneTalk: an expert exchange platform for assessing rare sequence variants in personal genomes. Bioinformatics, 28(19), 2515-2516.

Kesh, S. (1995). Evaluating the quality of entity relationship models. *Information and Software Technology*, *37*(12), 681-689.

Leavitt, N. (2010). Will NoSQL databases live up to their promise?. *Computer*, *43*(2), 12-14.

LinkedDataTools.com (2015). Introducing Linked Data And The Semantic Web [Web log post] Retrieved from http://www.linkeddatatools.com/

Machado, C. M., Rebholz-Schuhmann, D., Freitas, A. T., & Couto, F. M. (2015). The semantic web in translational medicine: current applications and future directions. *Briefings in bioinformatics*, *16*(1), 89-103.

Manifesto for Agile Software Development. 2001. [Web log post] Retrieved from http://www.agilemanifesto.org/

Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., ... & Sycara, K. (2004). OWL-S: Semantic markup for web services. *W3C member submission*, *22*, 2007-04.

Martone, M. (2012). Where do we go from here? (databases and ontologies). Retrieved from https://www.youtube.com/

Mazloumian, A., Eom, Y. H., Helbing, D., Lozano, S., & Fortunato, S. (2011). How citation boosts promote scientific paradigm shifts and nobel prizes. PloS one, 6(5), e18975.

McInerney, C., Bird, N., & Nucci, M. (2004). The flow of scientific knowledge from lab to the lay public the case of genetically modified food. Science Communication, 26(1), 44-74.

Meng, J., Mei, S., & Yan, Z. (2009, December). Restful web services: A solution for distributed data integration. In *Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on* (pp. 1-4). IEEE.

NCBI Bookshelf. (2014). Clustered RefSNPs (rs) and Other Data Computed in House [Web log post] Retrieved from http://www.ncbi.nlm.nih.gov/

O'Neill, O. (2003). Some limits of informed consent. *Journal of Medical Ethics*, *29*(1), 4-7.

O'Reilly, T. (2007). What is Web 2.0: Design patterns and business models for the next generation of software. *Communications & strategies*, (1), 17.

Ortega, J. L., & Aguillo, I. F. (2014). Microsoft academic search and Google scholar citations: Comparative analysis of author profiles. *Journal of the Association for Information Science and Technology*, *65*(6), 1149-1156.

Pautasso, C. (2014). RESTful web services: principles, patterns, emerging technologies. In *Web Services Foundations* (pp. 31-51). Springer New York.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, *12*, 2825-2830.

Peña, I., Cabezas, C., & Alonso, J. L. (2015). The Nucleoside Uridine Isolated in the Gas Phase. *Angewandte Chemie*, *127*(10), 3034-3037.

Phelps, L., Fox, B. A., & Marincola, F. M. (2012). Supporting the advancement of science: Open access publishing and the role of mandates. Journal of translational medicine, 10(1), 13.

Pokorny, J. (2013). NoSQL databases: a step to database scalability in web environment. *International Journal of Web Information Systems*, *9*(1), 69-82.

Poli, R., Healy, M., & Kameas, A. (Eds.). (2010). *Theory and Applications of Ontology: Computer Applications* (pp. 1-26). Dordrecht: Springer.

Ponzanni, G. (2013). Introduction to NoSQL. [PDF slides]. Retrieved from http://profs.sci.univr.it/

Price, W. N. (2015). Describing Black-Box Medicine. *Boston University journal of Science and Technology Law, Forthcoming*.

Raemaekers, S., Van Deursen, A., & Visser, J. (2014, September). Semantic versioning versus breaking changes: A study of the maven repository. In *Source Code Analysis and Manipulation (SCAM), 2014 IEEE 14th International Working Conference on* (pp. 215-224). IEEE.

Reeves, S., Kuper, A., & Hodges, B. D. (2008). Qualitative research methodologies: ethnography. *Bmj*, *337*.

Rindfleisch, T. C. (1997). Privacy, information technology, and health care. *Communications of the ACM*, *40*(8), 92-100.

Robbins, R. J. (1996). Bioinformatics: Essential Infrastructure for Global Biology1. *Journal of Computational Biology*, *3*(3), 465-478.

Ross, K. A., Janevski, A., & Stoyanovich, J. (2005, August). A faceted query engine applied to archaeology. In *Proceedings of the 31st international conference on Very large data bases* (pp. 1334-1337). VLDB Endowment.

Salem, D. N., & Boumil, M. M. (2013). Conflict of interest in open-access publishing. New England Journal of Medicine, 369(5), 491-491.

Sanders, R., & Kelly, D. (2008). Dealing with risk in scientific software development. IEEE software, 25(4), 21.

Scarano, D & Rao, R. (2014). DNA markers for food products authentication. Diversity, 6(3), 579–596.

Schulz, S., & Jansen, L. (2013). Formal ontologies in biomedical knowledge representation. *Yearb Med Inform*, *8*(1), 132-46.

Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, *34*(1), 1-47.

Shah, P. K., Perez-Iratxeta, C., Bork, P., & Andrade, M. A. (2003). Information extraction from full text scientific articles: Where are the keywords?. *BMC bioinformatics*, *4*(1), 20.

Sheridan, J., & Tennison, J. (2010). Linking UK Government Data. *LDOW*.

Smith, B. (2008, July). Ontology (Science). In *FOIS* (pp. 21-35).

Soldatova, L. N., & King, R. D. (2005). Are the current ontologies in biology good ontologies?. *Nature biotechnology*, *23*(9), 1095-1098.

Staab, S., & Studer, R. (Eds.). (2013). *Handbook on ontologies*. Springer Science & Business Media.

Stein, L. D. (2003). Integrating biological databases. Nature Reviews Genetics, 4(5), 337-345.

Stevens, R., Goble, C. A., & Bechhofer, S. (2000). Ontology-based knowledge representation for bioinformatics. Briefings in bioinformatics, 1(4), 398-414.

Stonebraker, M. (2010a). SQL databases v. NoSQL databases. *Communications of the ACM*, *53*(4), 10-11.

Stonebraker, M. (2010b). Errors in database systems, eventual consistency, and the cap theorem. *Communications of the ACM, BLOG@ ACM*.

Suber, P. (2004). *Open access overview*. Retrieved from http://www.planta.cn/

Suber, P. (2009). *A field guide to misunderstandings about open access*. Retrieved from http://legacy.earlham.edu/

The World Wide Web Consortium. (2015). How the Semantic Web Works [Web log post] Retrieved from http://www.w3.org/

Tsarkov, D., & Horrocks, I. (2006). FaCT++ description logic reasoner: System description. *Automated reasoning*, 292-297.

Tweed, R., & James, G. (2010). A universal nosql engine, using a tried and tested technology. *White Paper, Creative Commons Attribution CC-BY*.

US v. Swartz, 945 F. Supp. 2d 216 (D. Mass. 2013).

Venter, J. C., Adams, M. D., Myers, E. W., Li, P. W., Mural, R. J., Sutton, G. G., ... & Beasley, E. (2001). The sequence of the human genome. *science*, *291*(5507), 1304-1351.

Walters, W. H. (2009). Google Scholar search performance: Comparative recall and precision. *portal: Libraries and the Academy*, *9*(1), 5-24.

Weibel, S., Kunze, J., Lagoze, C., & Wolf, M. (1998). *Dublin core metadata for resource discovery* (No. RFC 2413).

Willinsky, J. (2005). The unacknowledged convergence of open source, open access, and open science. First Monday, 10(8).

Wirth, N. (1986). *Algorithms and data structures*. Upper Saddle River, New Jersey: Prentic Hall.

Yu, Q., Ding, Y., Song, M., Song, S., Liu, J., & Zhang, B. (2015). Tracing database usage: Detecting main paths in database link networks. *Journal of Informetrics*, *9*(1), 1-15.

Zou, D., Ma, L., Yu, J., & Zhang, Z. (2015). Biological Databases for Human Research. Genomics, proteomics & bioinformatics, 13(1), 55-63.

ADDENDUM

Script 1 in Python format

```
#Copies files containing potentially valid "rs" terms from source to destination directory
import os
import shutil
directory = "/home/werner/Desktop/Source/articles.O-Z/" #source directory
filelisting = os.walk(directory)
totalFiles = 0 #counter 1
rsFiles = 0 #counter 2
for root, dirs, files in filelisting:
    for file in files:
        totalFiles += 1
        breakTest = 0
        fileOne = open(root + "/" + file)
        if breakTest == 1:
            break
        for line in fileOne:
            line1 = line.split()
            if breakTest == 1:
                break
            for word in line1:
                #search clause
                if (word.startswith('rs') or (not word[0].isalnum() and "rs" in word[1:3])):
                    #destination directory
                    shutil.copy(os.path.join(root,file), "/home/werner/Desktop/Destination/" + file)
                    breakTest = 1
                    rsFiles += 1
                if breakTest == 1:
                    break
        fileOne.close()
print(totalFiles, rsFiles) #print ratio of files scanned to files containing rsSNPs
```

Script 2 in Python format

```
#Create flat file database
import xml.etree.ElementTree as ET
import os
import json
import unicodedata
def remove_control_characters(s): #Remove control characters in XML text
    t = ""
    for ch in s:
        if unicodedata.category(ch)[0] == "C":
            t += " "
        if ch == "," or ch == "\"":
            t += ""
        else:
            t += ch
    return "".join(ch for ch in t if unicodedata.category(ch)[0]!="C")
directory = "/home/werner/Desktop/Destination/"
filelisting = os.walk(directory)
rslist =[]
for root, dirs, files in filelisting:
    for file in files:
        email = "not available" #iteration reset
        pmid = "not available" #iteration reset
        year = "not available" #iteration reset
        doi = "not available" #iteration reset
        kwds = [] #iteration reset
        title = "" #iteration reset
        tree = ET.parse(root + "/" + file)
        #Get email addresses
        for node in tree.iter('email'):
            email = node.text
        #Get publication date
        for node in tree.iter('pub-date'):
            for subnode in node.iter('year'):
                collection = node.attrib
                if "pub-type" in collection.keys():
                    year = subnode.text
        #Get titles
        for node in tree.iter('title-group'):
            for subnode in node.iter('article-title'):
                whole = subnode.itertext()
                for parts in whole:
                    title += parts
        title = remove_control_characters(title)
        #Get PMC IDs
        for node in tree.iter('article-id'):
            pmidat = node.attrib
            if "pmc" in pmidat.values():
                pmid = node.text
            if "doi" in pmidat.values():
                doi = node.text
        #Get author defined keywords
        for node in tree.iter('kwd'):
            kwd = node.text
            if kwd != None:
                kwds.append(json.dumps(kwd))
        kwdstring = str(kwds)
        kwdstring = kwdstring.replace("'", "")
        kwdstring = kwdstring.replace("\\", "")
        #Get rs numbers
        for node in tree.iter():
            if node.text != None:
                node = node.text.split()
                for rsnumber in node:
                    #trim rs numbers preceded by opening bracket
                    if "rs" in rsnumber[1:3]:
                        rsnumber = rsnumber[1:]
                    #ensure that digits follow rs numbers and place limits on rs number length
                    if rsnumber.isalnum() and len(rsnumber) > 4 and len(rsnumber) <= 12:
                        #ensure that digits follow rs
                        if rsnumber.startswith("rs") and rsnumber[2].isdigit():
                            #ensure that rs numbers end with digits
                            while not rsnumber[-1].isdigit():
                                rsnumber = rsnumber[:-1]
                            #create list item without white spaces
                            rslist.append(rsnumber.strip() + "\t" + email.strip() + "\t" +
year.strip() + \
                            "\t" + pmid.strip() + "\t" + doi.strip() + "\t" + file.strip() + "\t" +
str(kwdstring) + "\t" + str(title))
#create refSNP occurence counter (with dictionary) to avoid data duplication
rsdict = {}
for item in rslist:
    #add data if counter is zero
    if rsdict.get(item,"empty") == "empty":
        rsdict.update({item:1})
    #increment counter if data was added
    if rsdict.get(item,"empty") != "empty":
        rsdict[item] += 1
rslist = []
#create tab delimited CSV file and add data
writetofile = open("/home/werner/Desktop/TEXTdb.csv", "a")
for item in rsdict.keys():
    writetofile.write(item + "\t" + str(rsdict[item]) + "\n")
writetofile.close()
```

Script 3 in Python format

```
#Convert tab delimited CSV file to JSON
import json
file = open("/home/werner/Desktop/TEXTdb.csv", "r")
JSONstring = "{\"PMCOAI_rs_articles\": ["
for line in file:
    lister = line.strip().split("\t")
    if len(lister) == 9:
        rs_number = "\"rs_number\": " + "\"" + lister[0] + "\""
        email_address = "\"email_address\": " + "\"" + lister[1] + "\""
        publication_date = "\"publication_date\": " + "\"" + lister[2] + "\""
        pubmed_id = "\"pubmed_id\": " + "\"" + lister[3] + "\""
        doi = "\"doi\": " + "\"" + lister[4] + "\""
        pubmed_file_name = "\"pubmed_file_name\": " + "\"" + lister[5] + "\""
        keywords = "\"keywords\": " + lister[6]
        article_title = "\"article_title\": " + "\"" + lister[7] + "\""
        rs_number_cited_in_article = "\"rs_number_cited_in_article\": " + "\"" + lister[8] + "\""
        newdictionary = "{" + rs_number + ", " + email_address + ", " + publication_date + ", " +
pubmed_id + \
        ", " + doi + ", " + pubmed_file_name + ", " + rs_number_cited_in_article + ", " +
article_title + ", " + keywords + "}"
        JSONstring += newdictionary + ", "
file.close()
#save JSON data and ensure last entry is enclosed in brackets
with open("/home/werner/Desktop/JSONdbKeys.json", "w") as file1:
    JSONstring = JSONstring[:-2]
    JSONstring += "]}"
    file1.write(JSONstring)
```

Script 4 in Python format

```
#Generate keyword list and count keywords in database
import json
keywordList = []
with open("/home/werner/Desktop/JSONdbKeys.json") as data_file:
    data = json.load(data_file)
    #Determine range to avoid out of bounds error
    for i in range(len(data["PMCOAI_rs_articles"])):
        if len(data["PMCOAI_rs_articles"][i]["keywords"]) > 0:
            for keywords in data["PMCOAI_rs_articles"][i]["keywords"]:
                keywords = keywords.strip()
                #Include alphanumeric terms and phrases
                if len(keywords) > 0 and all(x.isalnum() or x.isspace() for x in keywords) \
                and not keywords.startswith("rs"):
                    keywordList.append(keywords)
    #Remove duplicates with set
    keywordList = set(keywordList)
    #Revert to list
    keywordList = sorted(keywordList)
with open("/home/werner/Desktop/keywords.js", "w") as data_file:
    data_file.write(json.dumps(keywordList))
print(len(keywordList))
```

Script 5 in Python format

```
#Generate rs number list and count rs numbers in database
import json
keywordList = []
with open("/home/werner/Desktop/JSONdbKeys.json") as data_file:
    data = json.load(data_file)
    #Determine range to avoid out of bounds error
    for i in range(len(data["PMCOAI_rs_articles"])):
        #Include rs numbers with at least four digits following "rs"
        if len(data["PMCOAI_rs_articles"][i]["rs_number"]) > 5 and data["PMCOAI_rs_articles"][i]
["rs_number"][2:].isdigit():
            keywordList.append(data["PMCOAI_rs_articles"][i]["rs_number"])
    #Remove duplicates with set
    keywordList = set(keywordList)
    #Revert to list
    keywordList = sorted(keywordList)
with open("/home/werner/Desktop/rsnumbers.js", "w") as data_file:
    data_file.write(json.dumps(keywordList))
print(len(keywordList))
```

Script 6 in Javascript format

```
var jsonServer = require('../../json-server')
// Returns an Express server
var server = jsonServer.create()
// Set default middlewares (logger, static, cors and no-cache)
server.use(jsonServer.defaults)
// Create global file path variable
var multer = require('../../multer');
var upload = multer({ dest: 'uploads/' });
var mypath = "";
```

```
// Add custom routes
// Implement get route to accept VCF file uploads
server.post('/upload', upload.single('image'), function(req, res) {
             mypath = "";
       mypath += req.file.path;
       res.redirect("/confirmation.html")});
// Parse VCF file and return extracted SNPs to client
server.get('/extractSNP', function(req, res) {
    var awk = require('../../awk');
    var fs = require('fs');
    var intercept = require("../../intercept-stdout");
    var captured_text = "";
    //Initialize standard output hook
    var unhook_intercept = intercept(function(txt) {
    captured_text += txt;});
    awkscript = fs.readFileSync( __dirname + '/awkString.awk');
    data = fs.readFileSync(mypath);
    awk(awkscript, data);
    //Terminate standard output hook
    unhook_intercept();
    fs.writeFile('rsIDs.txt',captured_text,'utf8' ,function(){
        res.download('rsIDs.txt')});});
// Parse VCF file and return extracted genomic co-ordinates to client
server.get('/extractLoc', function(req, res) {
    var awk = require('../../awk');
    var fs = require('fs');
    var intercept = require("../../intercept-stdout");
    var captured_text = "";
    var unhook_intercept = intercept(function(txt) {
    captured_text += txt;
    });
    awkscript = fs.readFileSync( __dirname + '/awkString2.awk');
    data = fs.readFileSync(mypath);
    awk(awkscript, data);
    unhook_intercept();
    //Create callback to capture data before sending file to browser
    fs.writeFile('rsIDsPlus.txt',captured_text,'utf8' ,function(){
        res.download('rsIDsPlus.txt')});});
// Returns an Express router
var router = jsonServer.router('JSONdbKeys.json')
server.use(router)
server.listen(3000)
```

Script 7 in CSS format

```
/*Load default fonts*/
@font-face {
  font-family: 'ArchitectsDaughter';
  src: url(/fonts/ArchitectsDaughter.ttf);}
  @font-face {
  font-family: 'Comfortaa';
  src: url(/fonts/Comfortaa_Bold.ttf);}
/*Font overrides*/
.myFont {
  font-family: 'ArchitectsDaughter', cursive;
  text-shadow: 2px 2px black;}
* {
  font-family: 'Comfortaa', cursive;}
H1 {
  font-family: 'Comfortaa', cursive;
  text-shadow: 2px 2px black;}
H2 {
  font-family: 'Comfortaa', cursive;}
H3 {
  font-family: 'Comfortaa', cursive;}
H4 {
  font-family: 'Comfortaa', cursive;}
H5 {
  font-family: 'Comfortaa', cursive;}
H6 {
  font-family: 'Comfortaa', cursive;
}
/* Custom formatting */
.divspecs {
  margin: 40px 20px 40px 20px;
```

```css
  text-align: center;
  color: cornsilk;}
.updateSpecs {
  white-space: normal;
  margin-bottom: 10px;}
.formStyling {
  padding-bottom: 4px;}
.inputClass {
  margin: 0 auto;}
p {
  text-align: left;}
.centerClass {
  text-align: center;}
/*Page background*/
html, body {
  height: 100%;}
html {
  overflow-y: hidden;}
body {
  overflow-y: scroll;
  background-color:#000000;
  background: url(/images/DNA.jpg) no-repeat center center;
  background-size: 100% 100%;}
/*AJAX loader*/
.modal1 {
  display:    none;
  position:   fixed;
  z-index:    1000;
  top:        0;
  left:       0;
  height:     100%;
  width:      100%;
  background: rgba( 255, 255, 255, .8 )
  url(https://upload.wikimedia.org/wikipedia/commons/d/de/Ajax-loader.gif)
  50% 50%
  no-repeat;}
body.loading {
  overflow: hidden;}
body.loading .modal1 {
  display: block;}
/*Fivera image rotate*/
img {
  -webkit-transition: all 0.8s ease;
  -moz-transition: all 0.8s ease;
  -o-transition: all 0.8s ease;
  -ms-transition: all 0.8s ease;
  transition: all 0.8s ease;}
img:hover {
  -webkit-transform: rotate(20deg);
  -moz-transform: rotate(20deg);
  -o-transform: rotate(20deg);
  -ms-transform: rotate(20deg);
  transform: rotate(20deg);}
```

Script 8 in Javascript format

```javascript
//File upload event handler
function startRead() {
  var file = document.getElementById('file').files[0];
  if(file) {
    getAsText(file);};};
//Get file as text
function getAsText(readFile) {
  var reader;
  try {
    reader = new FileReader();
  } catch(e) {
    document.getElementById('output').innerHTML =
    "Error: seems File API is not supported on your browser";
    return;};
  // Read file into memory as UTF-8
  reader.readAsText(readFile, "UTF-8");
  // Handle progress, success, and errors
  reader.onload = loaded;
  reader.onerror = errorHandler;};
```

```
//Create filestring variable
var fileString = "";
//On load function
function loaded(evt) {
  fileString = evt.target.result;
  fileString = fileString.split("\n");
  for(var i=0; i < fileString.length; i++) {
    document.getElementById('output').innerHTML += (fileString[i] + "\n");};};
//On error function
function errorHandler(evt) {
  if(evt.target.error.code == evt.target.error.NOT_READABLE_ERR) {
    document.getElementById('output').innerHTML = "Error reading file..."}};
```