PharmGKB Accession and Bioconductor Integration

[*] Designed and implemented by MASAJ

Dr. Christopher Hemme
College of Pharmacy
University of Rhode Island
Kingston, Rhode Island
hemmecl@uri.edu

Michael Conti
Computer Science and Statistics
University of Rhode Island
Kingston, Rhode Island
michaelconti@uri.edu

Jacob Maloof
Computer Science and Statistics
University of Rhode Island
Kingston, Rhode Island
jacob_maloof@my.uri.edu

Alexander Ling
College of Pharmacy
University of Rhode Island
Kingston, Rhode Island
alexander_ling@my.uri.edu

Ali Amani
Computer Science and Statistics
University of Rhode Island
Kingston, Rhode Island
ali_amani@uri.edu

Sandy Chen
Cell and Molecular Biology
University of Rhode Island
Kingston, Rhode Island
sandy_chen@my.uri.edu

***Abstract*—Since 2001, PharmGKB has been providing clinical researchers with valuable information regarding gene-drug associations and genotype-phenotype relationships. For anyone interested in researching only one particular genome or drug, the database serves its purpose well. However, PharmGKB does not lend itself to one interested in performing large-scale statistical analysis or research on the families of genomes and drug relationships. Accessing large volumes of data from the database is neither quick nor efficient. For example, having this kind of data on hand could allow pharmacists/doctors to individualize treatments to their patients, maximizing therapeutic efficacy and safety. We seek to provide a solution to this problem by extracting the data into an easy-to-manipulate package within the R framework.**

***Index Terms*—PharmGKB, R-framework, Drug, Disease, Variant, Database**

I. Problem Definition

There exists many databases that have vast information on topics relating to diseases, drugs to those diseases, and variants of those drugs. Specifically, the PharmGKB database contains pharmacogenetic information about how variations in genotype can affect the response of drugs for a disease. However, the data on PharmGKB is not easily accessible for large scale data analysis. This is the problem we identified. We have created an R Package that accesses data related together in such a way that when a user inputs the name of a certain disease, the Package returns drugs that are made to treat the disease and the variants of the genotype that affect the drug. For example, inputting stroke should return drugs such as aspirin, warfarin, and clopidogrel. Then connected to warfarin should be variants of genes like CYP2C19 and VKORC1, and their effect on warfarin. Inversely, if the name of a drug is inputted, then the Package returns the diseases and the variants of the genotype. Our R Package allows for fast, easy access to information on related drugs, variants, and diseases, based on data inputted by the user.

Giving Bioconductor users this access allows for easy integration of the data available on PharmGKB into analysis pipelines. Rather than having to manually download or insert the data into their analysis process, we wish to give users the ability to automate the collection of this data. In the field of pharmacy, having large scale genomic information can allow pharmacists to predict what kind of reaction a population would likely have to the medications. This would help pharmacists to be aware that their patients may have certain side effects to the medications and how to quickly adjust the patients therapy. This can also help with hospitals and insurance companies develop formularies, as they would know which medications would be used more often because of the genetics of that population. Also having the genomic data of a population could help with drug discovery. As researchers learn more about how genetics play a role in diseases, they can also use those as targets for new drugs.

II. Background Introduction

Several databases already exists which offer a vast amount of information on drugs, drug variants, and diseases, and many other topics. However, we found in the API that we are using that the diseases in the database are not related to the drugs that treat them by the conventional method, meaning there is no direct field relation connecting diseases to drugs. Our R Package works around that problem by scanning a certain field in the table for drugs that has a small description, and in that description contains the name of the disease the drug treats. We found a shortcut that allows us to connect diseases with drugs by using what the API already has, improving upon it. Also, when using one of the currently existing databases, you stand the risk of having to wade through vast amounts of data before finally finding what you're looking for. Our R Package is only concerned with diseases, drugs, and variants,

With special thanks to Dr. Christopher Hemme for this guidance and support.

so the user can browse the data without having to go through information irrelevant to them, like they would with the larger databases. Our R Package improves upon what exists by avoiding information overload and narrowing down what is relevant to the user.

PharmGKB

PharmGKB is a public data repository for use in pharmacogenomic research. It contains gene and drug related data, most notably that concerning the relationships between drugs and their genomic variants. PharmGKB is widely used by both pharmacists and pharmacogenomic researchers. The majority of hits on the PharmGKB website are to look up the drug warfarin. The data can be accessed either through their website or by downloading it in an excel file, though there is currently no easy way to automate the accession and analysis of data on the website.

Bioconductor

Bioconductor is an open source R project for the development of genomic data analysis tools. It provides the user with a library of R packages designed for use in computational biology and bioinformatics research. Bioconductor is frequently used in the creation genomic data pipelines where users can automate analysis. A similar project for the Python programming language, called bioconda, is also widely used. However, our focus will be only on Bioconductor.

Bioconductor contains a few different packages with functions to download genomic data directly from an online databases. One such package is the aptly titled "GEOquery" package, last updated on April 16, 2019. This package allows for users to directly query the NCBI Gene Expression Omnibus, download the data of interest, and push that data into their analysis pipeline. The NCBI Gene Expression Omnibus is a public data repository containing information on genomic expression and other sequence-based data. Where as NCBI GEO contains sequence data, PharmGKB contains large volumes of data on the relationships between drugs, variants, and diseases.

Once finalized, our software will become a candidate for induction into Bioconductor for easy access and dissemination.

III. Software Requirements

Functional Requirements Definition

The R package will be usable within the Bioconductor framework in R. Using the package, the user will be able to query the PharmGKB database to search their term of choice (drug, gene, etc.) and obtain a result that compiles all related drugs, genes, and sequences. At a high level, our software will:

1) Pull data from PharmGKB
    a) Prompt user for drug name
    b) Make query to database for disease and variant, based on the entered drug
    c) If the query is successful, we display the data. Else, we return an error.

System Evolution

Maintenance of software is required to ensure consistency with PharmGKB API specifications. This is primaraly because to access the PharmGKB API, specific PharmGKB accession IDs a needed. Using our poster child example, Warfarin has a accession ID of 'PA166104776'. To ensure the developers (us) is speaking the same language as PharmGKB, we both need to refer to Warfarin as 'PA166104776'. As new drugs, diseases, and variants are discovered an updated dictionary of PharmGKB accession identifiers is required to access updated fields. Unfortunately, unless PharmGB changes the requirements for their API calls, an accession ID is required to retrieve a successful response from their database.

Hardware

Our software is not a resource hungry application, but the minimum hardware requirements should be observed to prevent unexpected behavior or low performance. The minimum and optimal requirements can be cited below:

1) Minimal Hardware Configurations
    a) 1 gigahertz (GHz) or faster 32-bit (x86) or 64-bit (x64) processor
    b) 1 gigabyte (GB) RAM (DDR3) or 2 GB RAM (DDR4)
    c) 16 GB available hard disk space (32-bit) or 20 GB (64-bit)
    d) Graphics Card - DisplayPort/HDMI or DVI support
    e) Internet access
2) Optimal Hardware Configurations
    a) Quad Core 2.4 GHz+ (i5 or i7 series Intel processor or equivalent AMD)
    b) 16 gigabyte (GB) RAM (DDR4)
    c) 256 GB or larger solid state hard drive
    d) Graphics Card - DisplayPort/HDMI or DVI support
    e) Internet access

IV. Results and Analysis

So far, our software has been able to send successful (200) queries to the PharmGKB server for a given drug or variant. With this accomplishment, we have been able to generate tree-like structures that link the relationship between a drug and it's given variants as expressed by PharmGKB.

While working on this project, we ran into implementation issues that we did not initially see as being problematic. That is, to reconstruct Figure 1.

It is a prerequisite that the drug, disease, and variant information is accessible and on the PharmGKB website for us to build the structure depicted in Figure 1. From an implementation perspective, we assumed that PharmGKB would structure their internal data such that there existed unique fields that held information on a given drug, disease, and variant. As we familiarize ourselves with the PharmGKB database, and to our detriment, we found that this was not the case. While there did exist specific and accessible fields for drug and variant information, the organization of the disease

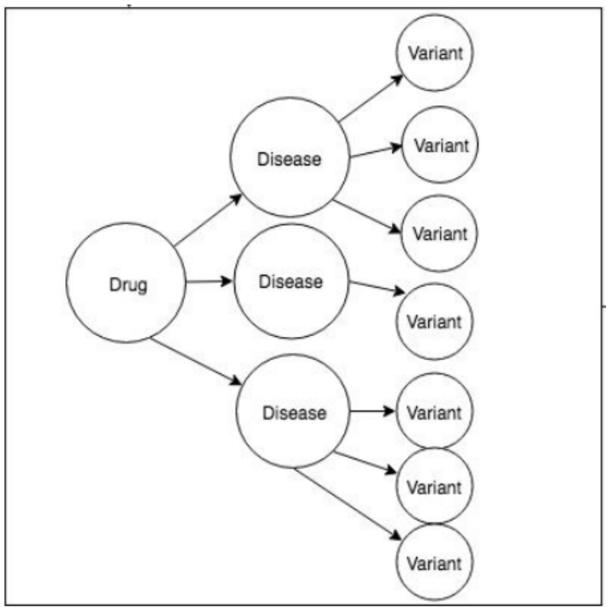

Fig. 1. Data structure representation

data was not what were were expecting and hoping for. To mitigate this constraint, we selected eight drugs Carvedilol, Abacavir, Metoprolol, Diazepam, Omeprazole, Valproic, Acid Phenytoin, Acetaminophen, and Warfarin, researched the diseases those drugs are used to treat, and defined a disease vector for a proof of concepts. That is, we implemented a temporary solution to the lack of disease data accessible from the PharmGKB database by locally storing disease data for a handful of commonly cited drugs.

To dive deeper into the structure of the PharmGKB database, we will describe where the disease information is stored. In the PharmGKB database, an annotation field exists for each variant. In this field, there are one of two cases for the value value in this field where case one is the value of the annotations field is null (or empty) and case two the value contains disease information in a sentence or paragraph format. This differs from our assumption that there exists a field or array of fields that contain only the disease name. If that were the case, it would be trivial to access that disease field, return the disease value, and fit that value into a child node of drug and parent node for variant. With PharmGKB's current representation of the disease field, we would need have a condition where we check the case of the annotations filed (null or !null) and in the case of where annotations is not null, read each character in a body of text of unknown length and pull out the disease name. The tricky part is that we would need to know what word would classify as a disease. A naive approach to this would be to cross check each word in the annotations text against an array of disease names, which at worst case would be linear (O(n)) for each word in our annotations text of length l so the final complexity would be O(n*l).

Another consideration that we did not make before our initial approach to representing our data is the explosive nature of variants for a given drug. That is, hundreds of variants could exist for one drug. Going back to Figure 1, this would result in a disproportionate number of leaf nodes in our tree. To mitigate this, the team made the design decision to return the first 10 variants for a given query unless a specific variant is searched.

One final workaround that we had to implement in our software is the inclusion of a variant and drug dictionary that is needed to return the PharmGKB accession identifier so we can query the PharmGKB database. To access the PharmGKB database for variant or drug information, a PharmGKB accession identifier is needed to query specific drugs/variants. Naturally, the user of this software would simply provide the name of a drug instead of a unique database identifier. However, because simply sending the name of the desired drug or variant won't work our software converts that string into the PharmGKB accession identifier and then uses that value to query the database.

Right now, our implementation status is considered incomplete because we still have some features that we outlined in our initial proposal that need to be added to our current software. Below are the specific features that have been implemented in out software so far:

A. *Software Architecture*

Our R software has two driving functions that interface with the PharmGKB API named **getVariant** and **getDrug**. These two functions accept a variant or drug as arguments (respectively), and consult a dictionary that returns the PharmGKB accession identifier. This PharmGKB accession identifier is used to construct a Client URL API (CURL) request. An example a successful PharmGKB client URL can be found below:

https://api.pharmgkb.org/v1/data/label/PA166104776

Once this URL is constructed, our R software depends on R packages httr and jsonlite to make a request to the PharmGKB API. If successful, a 200 response message along with a JSON object is returned to the software. If unsuccessful, and 404 response message is returned with an empty JSON object. The functions cleanStr is then used to clean out unwanted HTML characters and then rendered to the terminal. A summary of the necessary functions, objects, and data structures important to our software are listed below:

B. *Functions*

This function returns a JSON object containing data related to the user queried drug

getDrug(string drugName)

This function returns a JSON object containing data related to the user queried variant

getVariant(string variantName)

This function initiates the API call

GET()

This function cleans out unwanted HTML tags

cleanStr(string htmlString)

This function returns the named index of our drug/variant library

searchDrugByVariant(someVariant)

This function calls the drugDriver() function once the drug for the input variant is specified

getDrugByVariant()

This function calls all necessary functions if drug is primary search value

drugDriver()

This function dynamically builds tree nodes based on API response values

makeTree()

This function calls all of the other functions needed to run software

main()

C. Data Structures

- Objects

  This object contains the JSON response for the queried drug information.

  drugObj

  This object contains the JSON response for the queried variant information.

  variantObj
- Dictionaries

  This dictionary contains the key/value pair relationships between the drug names and the PharmGKB accession identifier.

  drugDic

  This dictionary contains the key/value pair relationships between the variant names and the PharmGKB accession identifier.

  variantDic

  This dictionary contains the drug/disease pair relationships.

  diseaseDic

The state diagram below shows, at a high level, how all of these functions and data structures interact to drive our desired output for the software.

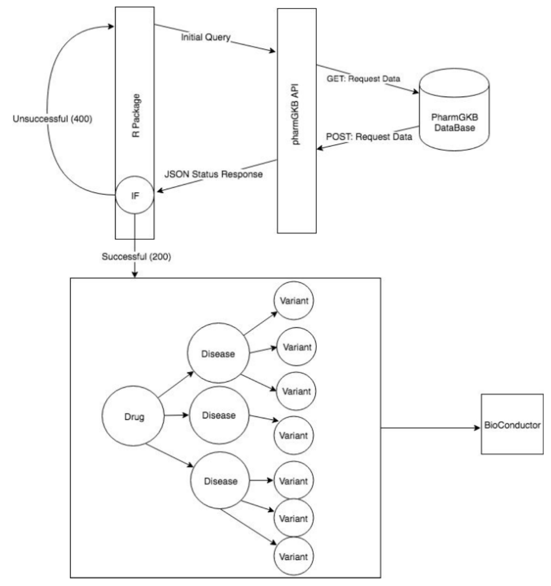

Fig. 2. Software Architecture State Diagram

To make these visualizations render in our software, we utilized the data.tree package in R. To create a graphic that shows the relationship between a drug, variant, and disease regardless of which drug is queried, we needed to dynamically create each level of of the tree based on the number of diseases and variants returned from the API call. We were able to accomplish this by looping through each element in the disease vector and create two child nodes for each disease - giving us a branching factor of b = 2 for this level. Additionally, we had to include a conditional to only make variant nodes while a variant value exists, guarding against the creation of null nodes. The ability to make tree structures dynamically based on the values in the disease and variant vectors allows the functionality of our software to scale to different drug inputs. An example output tree for the input 'warfarin' can be seen in Fig 3.

Words of wisdom for future developers

For those who plan to further this project in the future, we recommend that you take the time to develop a strong understanding of the structure of the PharmGKB database. This seemed to be our biggest downfall at the beginning of this process because we designed our approach and defined how we would interface with the PharmGKB database without truly knowing how it was set up, what was required to interface with it, and the format of the data that was actually accessible in the database. All of these things, while they might seem trivial, are critically important for the design and development of the software and method of approach to the PharmGKB database.

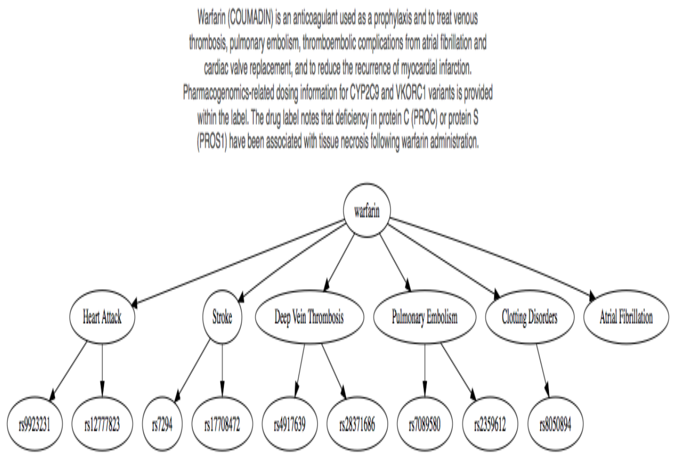
Warfarin (COUMADIN) is an anticoagulant used as a prophylaxis and to treat venous thrombosis, pulmonary embolism, thromboembolic complications from atrial fibrillation and cardiac valve replacement, and to reduce the recurrence of myocardial infarction. Pharmacogenomics-related dosing information for CYP2C9 and VKORC1 variants is provided within the label. The drug label notes that deficiency in protein C (PROC) or protein S (PROS1) have been associated with tissue necrosis following warfarin administration.

Fig. 3. R tree

D. *Conclusions and Future Work*

The greatest accomplished of this project, by far, is the successful navigation of the PharmGKB database to pull information into R. We believe the software that we have created is helpful for researchers that are interested in relationships between drugs and diseases as well as variants and drugs, as they no longer need to pull each one of these attributes one by one from the PharmGKB website.

We were not, however, able to pull all of the relationships between diseases and variants. For this to be possible, one would need either an encyclopedic understanding of the structure of the database, or we would need to perform a complete overhaul of the database itself. Please refer to Section III C to read a deeper outline of the database issues.

Overall, we were able to meet most of the goals we outlined in our initial project proposal. Further improvements could be made to this project by creating a more optimized and time-efficient process to citing the relationships between variants and diseases, rather than implementing the naive solution that we have come up with. With these improvements we believe the utility of our software would be significantly improved and therefore improving user experience.

Acknowledgment

A special thank you to Dr. Christopher Hemme, Dr. Noah Daniels, and Dr. Lenore Martin for their instruction and advice throughout this semester. This class has been a valuable learning experience for all of us in MASAJ and we would like to express our gratitude. We believe this class is a unique opportunity for University of Rhode Island students to participate in a cross functional and departmental teams which has allowed us all to develop in a personal and professional context. We hope this same opportunity is offered for future students for the years that come.

References

[1] Conti, Michael. CSC 522 Repo - R package for Pharmgkb. https://github.com/MichaelContii/csc522

[2] The Curl Package: a Modern R Interface to Libcurl, 9 Jan. 2019, cran.r-project.org/web/packages/curl/vignettes/intro.html.

[3] PharmGKB REST API, api.pharmgkb.org/swagger/.

[4] Kannan, Lavanya, et al. “Public Data and Open Source Tools for Multi-Assay Genomic Investigation of Disease.” Briefings in Bioinformatics, Oxford University Press, July 2016, www.ncbi.nlm.nih.gov/pmc/articles/PMC4945830/.

[5] “About.” Bioconductor, www.bioconductor.org/about/.

[6] “Home - GEO - NCBI.” National Center for Biotechnology Information, U.S. National Library of Medicine, www.ncbi.nlm.nih.gov/geo/.

[7] “GEOquery.” Bioconductor, www.bioconductor.org/packages/release /bioc/html/GEOquery.html.